ORIGINAL

Kimberly Caswell
Counsel

GTE SERVICE CORPORATION
One Tampa City Center
201 North Franklin Street (33602)
Post Office Box 110, FLTC0007
Tampa, Florida 33601-0110
813-483-2606
813-204-8870 (Facsimile)

April 17, 2000

Ms. Blanca S. Bayo, Director
Division of Records & Reporting
Florida Public Service Commission
2540 Shumard Oak Boulevard
Tallahassee, FL 32399-0850

Re: Docket No. 990649-TP
Investigation into Pricing of Unbundled Network Elements

Dear Ms. Bayo:

Please find enclosed for filing in the above matter an original and fifteen copies of GTE Florida Incorporated's Recurring Cost Study. This filing consists of a complete hard copy submission that includes the Model Methodology, User Guide, System Manual (with code), algorithms, input tables, model outputs and source documentation. Also included on CD-ROM is an electronic copy of GTE's cost model.

Also enclosed are an original and fifteen copies of a Notice of Intent to Seek Confidential Classification. All proprietary pages have been identified and marked as either Vendor Confidential or GTE Confidential. All confidential information is considered highly sensitive third-party proprietary information. Due to the high volume of paper, Parties of Record are not being served with redacted copies of the study. They will, however, receive the entire filing upon execution of the Protective Agreements which were mailed out on April 7, 2000.

Service has been made as indicated on the Certificate of Service. If there are any questions regarding this filing, please contact Elizabeth Florence in GTE's Cost & Pricing Group at (972) 718-4440.

AFA
APP
CAF
CMU 7
CTR
EAG
LEG 2
MAS 5
OPC
RRR
SEC 01
WAW
OTH

Sincerely,

Kimberly Caswell

KC:tas
Enclosures

* Binder 1, Tabs 1 thru 4 only. Copies of Binders 2 thru 15 returned to GTEFL to be corrected. Binder 1 and Redacted Version of Conf Version Binders 2 - 15 placed in org docket file. Law

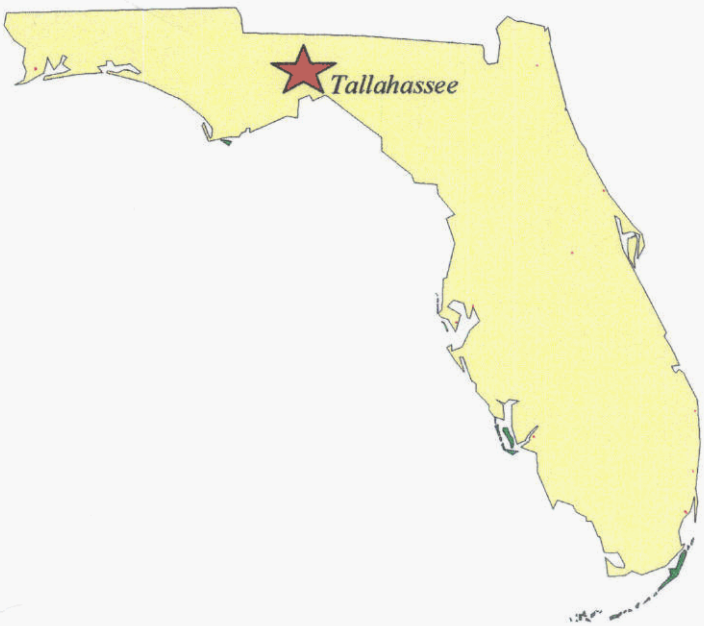
DNs 04615 thru 04629-00

ORIGINAL

GTE FLORIDA INCORPORATED

FLORIDA PUBLIC SERVICE COMMISSION

DOCKET NO. 990649-TP



Investigation Into Pricing Of Unbundled Network Elements

BINDER 1
TABS 1 - 4

APRIL 17, 2000

DOCUMENT NUMBER-DATE
04615 APR 17 8
FPSC-RECORDS/REPORTING



Integrated
Cost Model

Release 4.1

Model Methodology

Conceptual Framework

(Book I of VII)

GTE Service Corporation

Copyright © GTE 1997 - 2000. All rights reserved.

This is a Blank Page

ICM MODEL METHODOLOGY PACKAGE

➢ Conceptual Framework	Book I
➢ Loop Module	Book II
➢ Switch Module	Book III
➢ Interoffice Transport Module	Book IV
➢ SS7 Module	Book V
➢ Expense Module	Book VI
➢ Mapping/Report Module	Book VII

This is a Blank Page

Table of Contents

ICM MODEL METHODOLOGY PACKAGE ..... **a**

PREFACE ..... **iii**

ICM OVERVIEW **1**

Introduction 1

Model Design 2

Data Inputs 4

User Options 5

Reports 5

Software/Computer Requirements 6

APPENDIX A: Data Input Tables **7**

User Variable Inputs 7

Associate Table [USETABLE.DB] 14

Processing Flow 15

APPENDIX B: Glossary **16**

This is a Blank Page

PREFACE

This document presents an overview of the methodology used to develop GTE's Integrated Cost Model (ICM) version 4.1. The ICM Model Methodology is divided into seven sections: Conceptual Framework (Book I); Loop Module (Book II); Switch Module (Book III); Interoffice Transport Module (Book IV); SS7 Module (Book V); Expense Module (Book VI); and Mapping/Report Module (Book VII). Appendices containing more detailed information are also provided in each booklet.

In addition to these seven booklets, a separate *ICM User Guide* has been developed that explains how to install ICM, run reports, view data, and perform sensitivity analyses. The *System Manual* is divided in two sections: ICM Database Tables (Book I) and ICM System Codes (Book II). Both documents are attached.

This is a Blank Page

ICM OVERVIEW

Introduction

The Telecommunications Act of 1996 resulted in an increased demand for cost studies in both state and federal jurisdictions. For GTE, with telephone operations in 28 states, this presented a particularly difficult challenge.

Prior models, originally designed to perform service cost studies, had to be adapted to meet new cost study requirements. GTE's prior cost study process involved the running of several models, combined and tailored by cost study analysts to address particular service and cost study questions. Many of the input prices for material and labor were entered in the model's code and could only be reviewed by those with extensive knowledge of GTE's models and vendor relationships. Additionally, access to valuable information regarding commercially available models was limited due to vendor non-disclosure agreements.

This reality made it difficult to readily respond to cost study demands, including sensitivity analyses, for a multitude of state jurisdictions. The process also required extensive time and effort by GTE's cost study experts to train external parties on the details and results of each of the models.

As a result, GTE clearly needed to develop a single user-friendly model that would enable parties to more easily review and understand the company's cost studies. An integrated cost model would enhance GTE's ability to respond to cost study demands within the time frames dictated by the Telecommunications Act of 1996.

GTE developed the Integrated Cost Model (ICM) to address these needs. ICM integrates modules for each of the major network components and therefore streamlines the cost study process. With ICM, cost studies are presented clearly and interested parties are able to perform sensitivity analyses by changing various inputs. The PC-based model is designed to be open and accessible.

ICM reflects GTE's engineering standards, the technologies GTE is using now and going forward, and GTE's data and network topology. The resulting cost estimates represent the forward-looking costs for GTE.

ICM version 4.1 calculates cost estimates for GTE's basic network functions (BNFs), unbundled network elements (UNEs), and universal service. It includes costs related to the loop, switching, transport and signaling portions of the network. Service results are also developed although the final results of

the usage components are calculated external to ICM in SAS[1] and other miscellaneous studies.

Model Design

ICM has been designed to streamline the cost study process within GTE and make the company's cost studies and results open and user-friendly. The model features an easy-to-use, Windows-based user interface. Cost study inputs, intermediate results, and outputs are presented in tables that can be viewed or printed. Additionally, these tables can be extracted for use in such analytical tool as spreadsheets and database programs. The model develops results efficiently on personal computers typically used in business offices. Most of the key inputs can be varied easily by the user, allowing rapid sensitivity analyses. (See "User Options" below.)

ICM develops costs estimates related to engineering, installing, and maintaining a modern digital network required to provide efficient telephone service in GTE's service areas. GTE's planning guidelines were used to develop the model methodology. Economically efficient, forward-looking technology is modeled. For example, the loop is designed to use fiber optic cable and pair gain devices, where appropriate, and only digital switching technology is used.

A basic flow chart of ICM is shown in Figure 1.

[1]SAS is a commercially-available software package developed by the SAS Institute, Inc. for data warehousing, statistical analysis, and report generation.

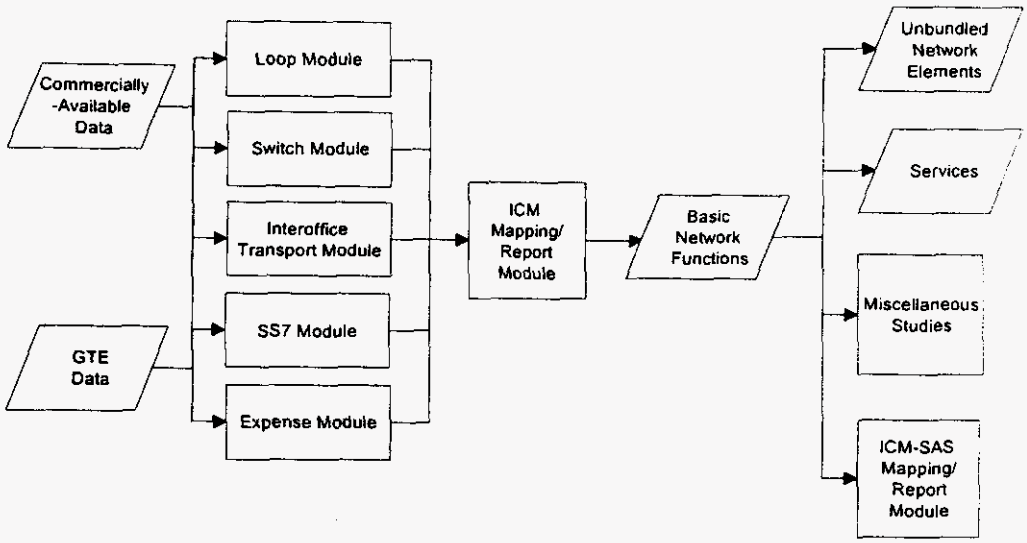


Figure 1: ICM Flow Chart

The Loop, Switch, Interoffice Transport and Signaling System 7 (SS7) Modules develop the investments for network components and the Expense Module identifies non-capitalized labor costs such as billing, repair and service assurance. The Mapping/Report Module calculates the monthly expenses, combines the network components into unbundled network elements and services, and produces output reports.

The Loop Module develops investments for network interface devices (NIDs), drops, distribution cable, terminals, cross connect boxes, pair gain devices, feeder cable, and related support structures.

The Switch Module develops the investment for setup and minutes of use for each type of call path, for switched features, and line and trunk terminations.

The Interoffice Transport Module (or simply, the "Transport Module") develops the investment for the outside plant facilities that connect switches and the specialized transmission equipment within wire centers. Signaling investments are determined in the SS7 Module.

Figure 2 below depicts many of the network components modeled in ICM. See the Loop, Switch, Transport and SS7 Module documentation for more complete descriptions of the investment components developed.

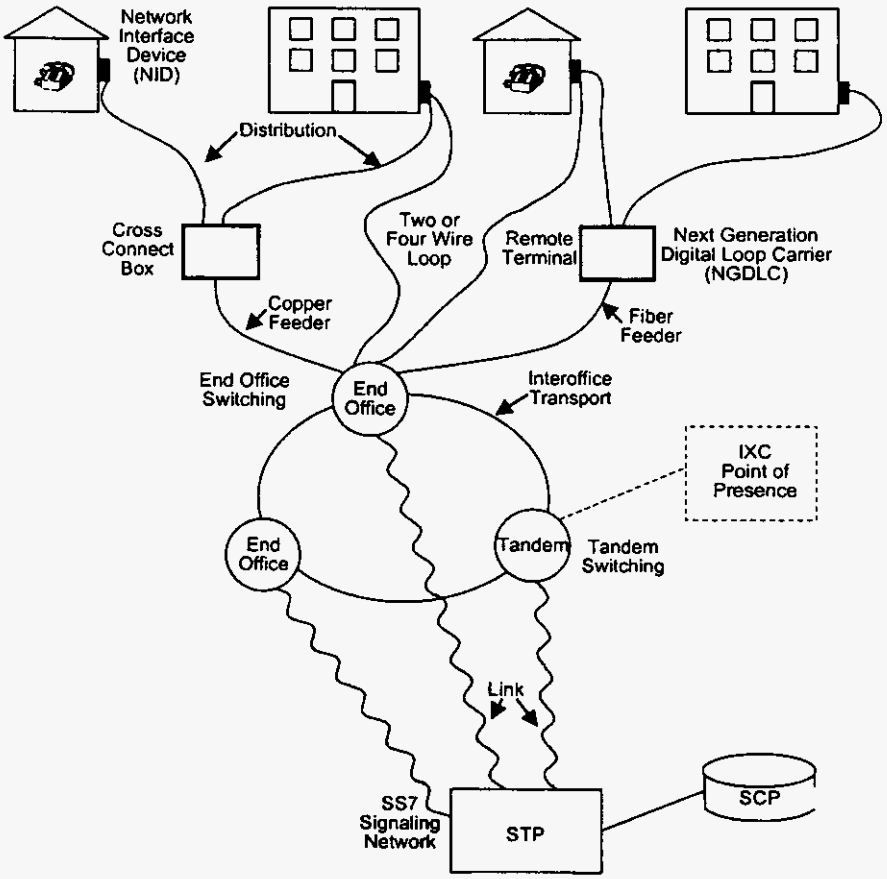


Figure 2: Network Components

Data Inputs

ICM data inputs are stored in tables that can be easily modified, viewed, and printed. Appendix A of each module booklet contains a detailed description of each input table related to that particular module. Some examples of data inputs that may be modified include:

- Material and placement cost
- Demand data
- Switch types
- Interoffice ring configurations

The user can vary inputs within ICM or by exporting a data input table, making modifications in another program (e.g., Excel or Lotus), and then importing the table back into the model. Data inputs can also be added to a

table; for example, a material price for cable of a particular size can be inserted in the outside plant material table[2]

ICM uses GTE-specific data such as switch locations, switch types, tariff boundaries, material prices, labor activity rates, ARMIS[3] data, and capital structure. All core materials are purchased centrally and a single set of base costs is input for all states. These costs reflect GTE's volume purchase agreements. The model then applies loadings, such as sales tax, shipping, minor materials and installation that are state specific.

In addition, Stopwatch Maps, Inc.[4], in conjunction with PNR and Associates, Inc.[5], provides data by geographic location including road footage, terrain conditions, and working line estimates. Appendix B of the Loop Module Book II within the ICM Methodology package includes a detailed description of the data inputs furnished by Stopwatch/PNR.

User Options

ICM allows the user to modify the inputs found under Options/User (such as the amount of structure sharing or copper loop length). The *ICM User Guide* provides instructions for changing these inputs to perform sensitivity analyses. ICM is designed to minimize processing time to allow for quick analysis of the impact of changes. The model will determine the data inputs that have been changed and only re-run the applicable modules.

Reports

The Mapping/Report Module produces detailed output reports for basic network functions (BNFs), Total Element Long Run Incremental Costs (TELRICs) and Total Service Long Run Incremental Costs (TSLRICs). ICM provides total costs for network functions and elements but the results for services do not include usage components. Usage BNF outputs for services are passed to the ICM-SAS Mapping/Report Module where TSLRICs for switched usage, (e.g., local usage, Toll Extended Calling Service or Message Telecommunications Service) are developed.

Results are available at both the state and wire center level, where appropriate. In ICM, reports can be both previewed in the model and printed for review. The report formats are more completely described in the Mapping/Report Module documentation. Also, Appendix A of the Mapping/Report Module documentation, under the subheading 'Processing Flow', presents a graphical view of the tables processed in ICM to create the model results.

[2] Note: Adding a material input might require the addition of a corresponding labor activity input.
[3] Automated Reporting Management Information System
[4] Stopwatch Maps, Inc., is a commercial firm that supplies detailed geographical data and formats data supplied by others.
[5] PNR Associates is a commercial firm that supplies estimates of residence and business access lines by census block.

Software/Computer Requirements

ICM has been developed using:

Delphi™ - an object-oriented Pascal programming language

Paradox™ - a database software application

The program is provided in a CD format.

To operate ICM, the user's computer must meet the following minimum requirements:

IBM-compatible PC

Pentium™ 120 - MHz microprocessor

16 MB RAM

1 GB available hard drive space

Windows 95 or NT™

CD drive

For information on how to install and operate ICM, see the *ICM User Guide*.

To produce some of the switched-usage TSLRICs such as message toll service and local measured service, ICM outputs must be imported into a SAS™ program that maps the appropriate usage components and develops the final results for services. When applicable, the ICM-SAS file is provided with ICM and the SAS application is required to perform this function.

™ All trademarks are the property of their respective owners.

APPENDIX A: Data Input Tables

This Appendix is divided into three sections: (I) A detailed description of User Variable Inputs; (II) Associate Table; and (III) Processing Flow Overview Diagram.

Part I provides descriptions of the ICM inputs that the user may vary to perform sensitivity analyses. To vary these inputs in ICM, click Options/User and select Outside Plant, Interoffice, or Expense. The Outside Plant Tab is divided into three parts: General, Distribution, and Feeder. Note: The user may vary these inputs as well as the data in the tables outlined in Appendix A of each module.

User Variable Inputs

OUTSIDE PLANT-GENERAL USER OPTIONS (ROOSP.DB)	**DESCRIPTION**
Pole Spacing	The typical number of feet between poles.
Manhole Spacing	The typical number of feet between manholes.
Manhole rock rem. fact	Manhole Rock Removal Factor. Number of pole holes in the bedrock that would be required to remove the bedrock in the area when placing a manhole
Pull Box Spacing	The typical number of feet between pull boxes.
Pull Box rock rem fact	Pull Box Rock Removal Factor. Number of pole holes in the bedrock that would be required to remove the bedrock in the area used as a surrogate to placing a pull box
Well Point Days	The number of days that well points would be required to remove ground water in the area excavated for manhole placement.
Percent Boring	The percent of the buried trench that requires boring (tunneling) under pavement or other obstructions.
Percent Guy Wire	Decimal representing the portion of poles which require anchors and down guys.

OUTSIDE PLANT-GENERAL USER OPTIONS (ROOSP.DB)	DESCRIPTION
Percent Concrete	The percent of buried trench that requires removing and replacing concrete or asphalt pavement.
Percent Hand Dig	Decimal representing percentage of trench that requires hand digging in lieu of mechanized trenching. This hand digging is in addition to the incidental hand digging included in the trenching costs.
Copper Loop Length	Allows the user to select a 12,000 kft, 12,000 kft 6Mbps or 18,000 kft maximum copper loop length
Administrative Fill Factor	Decimal representing the percentage of copper cable capacity that could be used for working lines, accounting for defective and administrative pairs.
Planning Horizon	Currently Not Applicable
Growth Rate	Currently Not Applicable
Users - Poles	The number of companies, including GTE, which are sharing the shared poles.
Users - Trench	The number of companies, including GTE, which are sharing the trench.
Users - Additional Conduit	The number of ducts placed in shared conduit for companies other than GTE.
Spans - Aerial Copper (Cu)	The typical number of feet between aerial splices in copper cables.
Spans - Buried Copper (Cu)	The typical number of feet between buried splices in copper cables.
Spans - Aerial Fiber (Fi)	The typical number of feet between aerial splices in fiber feeder cables.
Spans - Buried Fiber (Fi)	The typical number of feet between buried splices in fiber feeder cables.
Use User Fill	Allows the user to input utilization factors for distribution and feeder. The Loop Module adjusts cable investments to reflect the specified fill levels.

OUTSIDE PLANT-DISTRIBUTION USER OPTIONS (ROOSP.DB)	DESCRIPTION
Drop Size	Number of pairs contained in the service (drop) wire placed from the distribution serving terminal to the NID at the customer's location (User selects from either 3 or 5).
Maximum Length Drop Wire	The maximum length drop wire
Minimum Length Drop Wire	The minimum length drop wire
Business Threshold	Number of business lines in a demand unit above which the drop size for business lines is set to the lower Business Pair size.
Business Pairs	Drop size utilized for business lines when the number of business lines in a demand unit is between the two Business Thresholds. Typically set to 25 pairs.
Business Threshold	Number of business lines in a demand unit above which the drop size for business lines is set to the higher Business Pair size.
Business Pairs	Drop size utilized for business lines when the number of business lines in a demand unit exceeds the upper Business Threshold. Typically set to 50 pairs.
Residence Threshold	Number of residential lines in a demand unit above which the drop size for residential lines is set to the Residence Pair size.
Residence Pairs	Drop size utilized for residence lines when the number of residence lines in a demand unit exceeds the Residence Threshold. Typically set to 25 pairs.
Units Lines/Res	Ratio of number of residential lines, to the number of residential locations within a demand unit.
Units Lines/Bus	Ratio of number of business lines, to the number of business locations within a demand unit.
Terminals: Res/Units	The number of residential locations that are served from each terminal. Applies to residence locations utilizing 3 or 5 pair drops.

OUTSIDE PLANT-DISTRIBUTION USER OPTIONS (ROOSP.DB)	**DESCRIPTION**
Terminals: Bus/Units	The number of business locations that are served from each terminal. Applies to business locations utilizing 3 or 5 pair drops.
Sharing Distribution Percent - Foreign Poles	Decimal representing the portion of distribution poles that are not owned by GTE, which GTE attaches to through lease arrangements.
Sharing Distribution Percent - Aerial	Percent of distribution poles owned by GTE, and shared with others companies. (For the quantity of others see Users/Poles)
Sharing Distribution Percent - Buried	Percent of distribution trench owned by GTE, and shared with other companies. (For the quantity of others see Users/Trench)
Sharing Distribution Percent - Underground	Percent factor of distribution conduit owned by GTE, and shared with other companies.
Percent of no cost Dist. Trench	Represents the percent of distribution cable trenching provided by developers at no cost to GTE (in states where applicable).
Percent of no cost Drop Placement	Percent of Residential drops installed by developers at no cost to GTE (in states where applicable)
Engineering Distribution Factor	Factor applied to working lines to properly size distribution cable.

OUTSIDE PLANT-FEEDER USER OPTIONS (ROOSP.DB)	DESCRIPTION
Sharing Feeder Percent - Foreign Poles	Decimal representing feeder poles owned by other companies to which GTE attaches through lease arrangements.
Sharing Feeder Percent - Aerial	Percent of feeder poles owned by GTE and shared with other companies
Sharing Feeder Percent - Buried	Percent of feeder trench owned by GTE and shared with other companies
Sharing Feeder Percent - Underground	Percent of feeder conduit owned by GTE and shared with other companies
Cross Connect Box Factor	Factor is applied to the demand used to size the feeder cable to determine the size of a secondary cross connect box. It is based on the rule of "one pair in and two pairs out", which means for every feeder pair entering the cross connect box there are typically two distribution pairs leaving the box.
Min Cross Connect Size 1	User input demand level that triggers placement of a primary cross connect box (SAI, serving area interface). In non-core clusters, the primary SAI are placed adjacent to the Digital Loop Carrier (DLC), at the center of the cluster. Primary SAIs are not placed in core clusters.
Min Cross Connect Size 2	User input demand level that triggers placement of a secondary cross connect box (SAI, serving area interface). These are used in core and non-core clusters. In the core and non-core clusters, Secondary SAIs are placed along a route(s) away from the center of the cluster.
Minimum Distance	Minimum distance from a Digital Loop Carrier (DLC) or Wire Center to a secondary cross-connect box
Engineering Feeder Factor	Factor applied to working lines to properly size feeder cable and DLC equipment.

INTEROFFICE USER OPTIONS (IOFSET.DB)	DESCRIPTION
Administrative Fill Factor	Decimal representing percentage of interoffice facilities available for actual traffic; accounts for maintenance, spares, and defective material.
Intra-Ring Factor	Reflects the percentage of traffic that originates and terminates on the same ring. Used to calculate the total switched traffic per ring.
Aerial Span	The typical number of feet between aerial splices in fiber facilities.
Buried Span	Represents the typical number of feet between buried splices in fiber facilities.
Air to Route Ratio	Factor which converts airline distance to route distance (The factor is entered as route distance divided by airline distance and will thus be greater than one, e.g., 1.37)

EXPENSE USER OPTIONS (ROEXPNSE.DB)	DESCRIPTION
Life (Book or Economic)	Toggle which indicates whether Book or Economic depreciation lives and salvage values are applied to each asset's investment.
Market (Retail or Wholesale)	Toggle which indicates whether retail or wholesale expenses are applied. Retail is selected when results are being produced for products and services. Wholesale is selected when results are being produced for unbundled network elements (UNEs).
Shared	Shared expenses are included in the individual cost pools and the calculation of the maintenance and support factors for UNEs within ICM if Shared is checked. When Shared is not checked, shared expenses are excluded from cost pools and the calculation of the maintenance and support factors within ICM.
Inflation	Inflation factor applied to expenses.
Productivity	Productivity factor applied to expenses.
Horizon	A time frame (number of years) over which a user decides to make inflation and productivity adjustments.
Calibrate	Toggle indicates whether ICM should automatically calibrate the C.A. Turner adjusted ARMIS investment data in ICM associated with Switch, Circuit Equipment and Outside Plant activity to the levels calculated by the Switch, Loop, Interoffice Transport and SS7 modules of ICM. A checkmark in this field indicates the computation of calibration factors with the resulting factors appearing in the Adjust3 column of the XXGTEEXP database file for the following accounts: **Category:** **FCC Part 32 account(s):** Switching 2212 Circuit Equipment 2232 Outside Plant (OSP) 2411 through 2441, excluding 2431

Associate Table [USETABLE.DB]

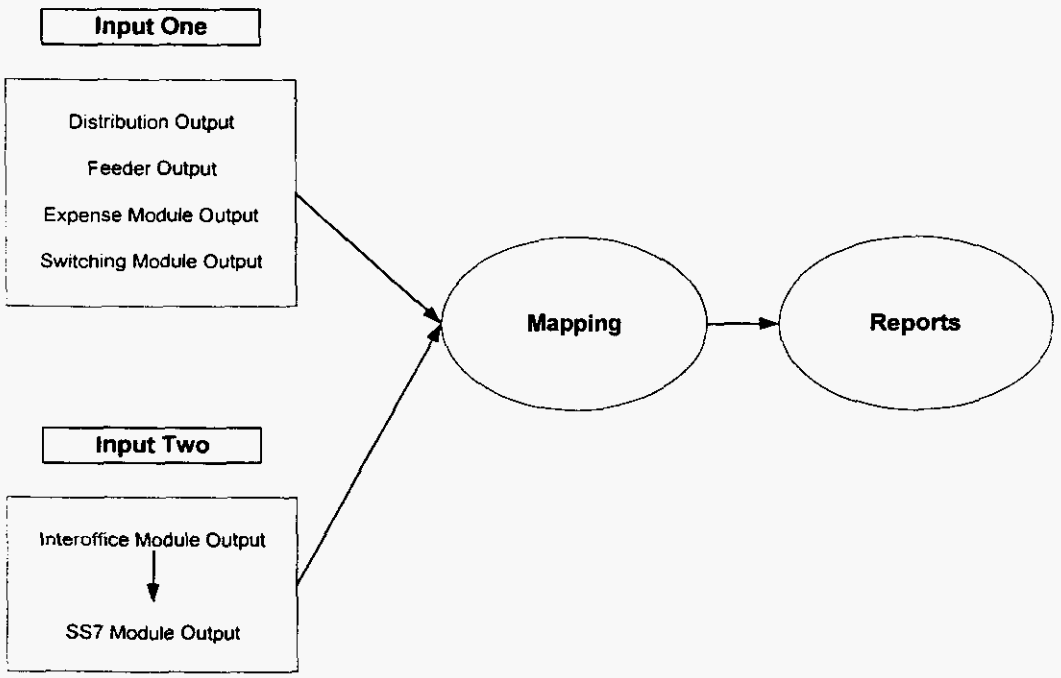
Purpose: Allows the user to indicate the location of the data files to be used by ICM.

COLUMN HEADINGS	DESCRIPTION
LOOKUP TABLE	Gives the names of the data input or mapping tables (e.g., Switches or Grid Squares)
FILENAME	Path and file name of the lookup table (e.g., ICM\DATABASE\SWITCH.DB)

Processing Flow

The diagram below represents all of the processing flow diagrams provided in each module's Appendix A. Below is a graphic view of the database files that are input to, and output of, each Module. For more information, refer to Appendix A of each module in the ICM Methodology package.

Integrated Cost Model



* In Input Two, the output of Interoffice Module is an input in the SS7 Module.

APPENDIX B: Glossary

Acronym	Description
A Link	Access Link
ADM	Add/Drop Multiplexer
ANSI	American National Standard Institute
ARMIS	Automated Reporting Management Information System
B Link	Bridge Link
BNF	Basic Network Functions
BRI	Basic Rate Interface
C Link	Cross Link
CCS	Centum Call Seconds
CLLI	Common Language Location Identifier
CMST	Constrained Minimum Spanning Tree
COT	Central Office Terminal
D Link	Diagonal Link
DCS	Digital Cross-Connect System
DLC	Digital Loop Carrier
EF&I	Engineered, Furnished and Installed
ESA	Electronic Serving Area
GTEAMS	GTE Advance Materials System
ICM	Integrated Cost Model
ISDN	Integrated Services Digital Network
LIBD	Line Information Database
MACRS	Modified Accelerated Capital Recovery System
MDF	Main Distribution Frame
MOU	Minute of Use
NECA	National Exchange Carrier Association
NGDLC	Next Generation Digital Loop Carrier
NID	Network Interface Device

Acronym	Description
OSP	Outside Plant
PBX	Private Branch Exchange
POP	Point of Presence
PRI	Primary Rate Interface
PSTN	Public Switched Telephone Network
RT	Remote Terminal
RTU	Right To Use
SCIS	Switching Cost Information System
SCP	Service Control Point
SONET	Synchronous Optical Network
SP	Signaling Point
SS7	Signaling System 7
SSP	Service Switching Point
STP	Signal Transfer Point
TELRIC	Total Element Long Run Incremental Cost
TIGER	Topologically Integrated Geographic Encoding and Referencing
TSLRIC	Total Service Long Run Incremental Cost
UNE	Unbundled Network Element
USOA	Uniform System of Accounts

This is a Blank Page



Integrated
Cost Model

Release 4.1

Model Methodology

Loop Module

(Book II of VII)

GTE Service Corporation

Copyright © GTE 1997 - 2000. All rights reserved.

This is a Blank Page

ICM MODEL METHODOLOGY PACKAGE

- Conceptual Framework — Book I
- Loop Module — Book II
- Switch Module — Book III
- Interoffice Transport Module — Book IV
- SS7 Module — Book V
- Expense Module — Book VI
- Mapping/Report Module — Book VII

This is a Blank Page

Table of Contents

ICM MODEL METHODOLOGY PACKAGE ..... **a**

INTRODUCTION **1**
Local Loop 1
Summary Of Features 1

MODEL METHODOLOGY **3**
Loop Module Design 3
Uniform Demand Unit 3
Electronic Serving Area (ESA) Development 7
Local Loop Network Design 8
Detailed Engineering 12
Facility Investment Development 12
Support Structure Investment Development 17
Distribution Template Design 24
Cross Connect (SAI) Development 25
Engineering Factors 27
Average Loop Length Calculation 27
Loop Module Outputs 28

CONCLUSION **29**

APPENDIX A: Data Input Tables **30**
Demand Table [XXDEMAND.DB] 31
Distribution Style Table [XXTEMPLT.DB] 32
Outside Plant Material [XXMATL.DB] 34
OSP Material Input Descriptions 35
Outside Plant Placement Costs [XXLABR.DB] 47
Outside Plant Placement Input Descriptions 48

Processing Flow....59

APPENDIX B: Stopwatch Maps' Role In Generating Demand Unit File....61

APPENDIX C: Distribution Styles & Distribution Diagrams....65

Stick Map for Distribution Style One....66

Stick Map for Distribution Style Two....67

Stick Map for Distribution Style Three....68

Stick Map for Distribution Style Four....69

Stick Map for Distribution Style Five....70

Stick Map for Distribution Style Six....71

Stick Map for Distribution Style Seven....72

Stick Map for Distribution Style Eight....73

Stick Map for Distribution Style Nine....74

APPENDIX D: Loop Components....75

Distribution Components....76

Feeder Components....77

INTRODUCTION

Local Loop

The Loop Module is designed to develop investments for local loops that are capable of providing advanced digital services. These loops can be used for single-line residential and business services, Key lines, PBX trunks, Centranet lines, as well as Integrated Service Digital Network (ISDN) and Asymmetric Digital Subscriber Line (ADSL). The module develops volume sensitive and volume insensitive investments for each of the loop components between the protector at the customer's premises and the main distribution frame in the central office (see Appendix D).

A table containing component names and related investments developed by the Loop Module is exported to the Mapping Module (refer to Book VII in the Integrated Cost Model (ICM) Methodology package for details[1]). In the Mapping Module, product and services costs are developed based on inputs from other modules, and cost definitions provided by the user. These cost definitions establish how the components developed in the Loop, Expense, Interoffice Transport and Switch modules are to be assembled into product and services.

Summary Of Features

At the core of ICM 4.1 are a pair of primary algorithms that allow the loop module to dynamically determine the location of pair gain devices, the network connecting those devices to the wire center, and the cells within the Electronic Serving Area (ESA). An ESA is an area in which all customers have access to a loop capable of providing digital services. The first algorithm is a K-mean cluster algorithm. The second algorithm is a constrained Minimum Spanning Tree (CMST) algorithm. These two algorithms permit the rapid development of detailed network investments.

Supporting the primary algorithms are a set of detailed engineering assumptions and user inputs that permit the user to determine, on a step-by-step basis, how each investment item is calculated and to perform sensitivity analysis at each stage.

Even with all of the additional detail that has been built into ICM 4.1, reasonable run-time still remains one of the key features that was common to earlier versions. To facilitate the sensitivity analysis permitted by relatively

[1] The ICM Methodology package is divided into seven booklets: Conceptual Framework (Book I); Loop Module (Book II); Switch Module (Book III); Interoffice Transport Module (Book IV); SS7 Module (Book V); Expense Module (Book VI); and Mapping/Report Module (Book VII).

quick run-times, a graphical interface allows the user to view the network being developed as well as to analyze the output. The basic flow chart below gives an overview of the general modeling approach and shows how investments for loop components are developed.

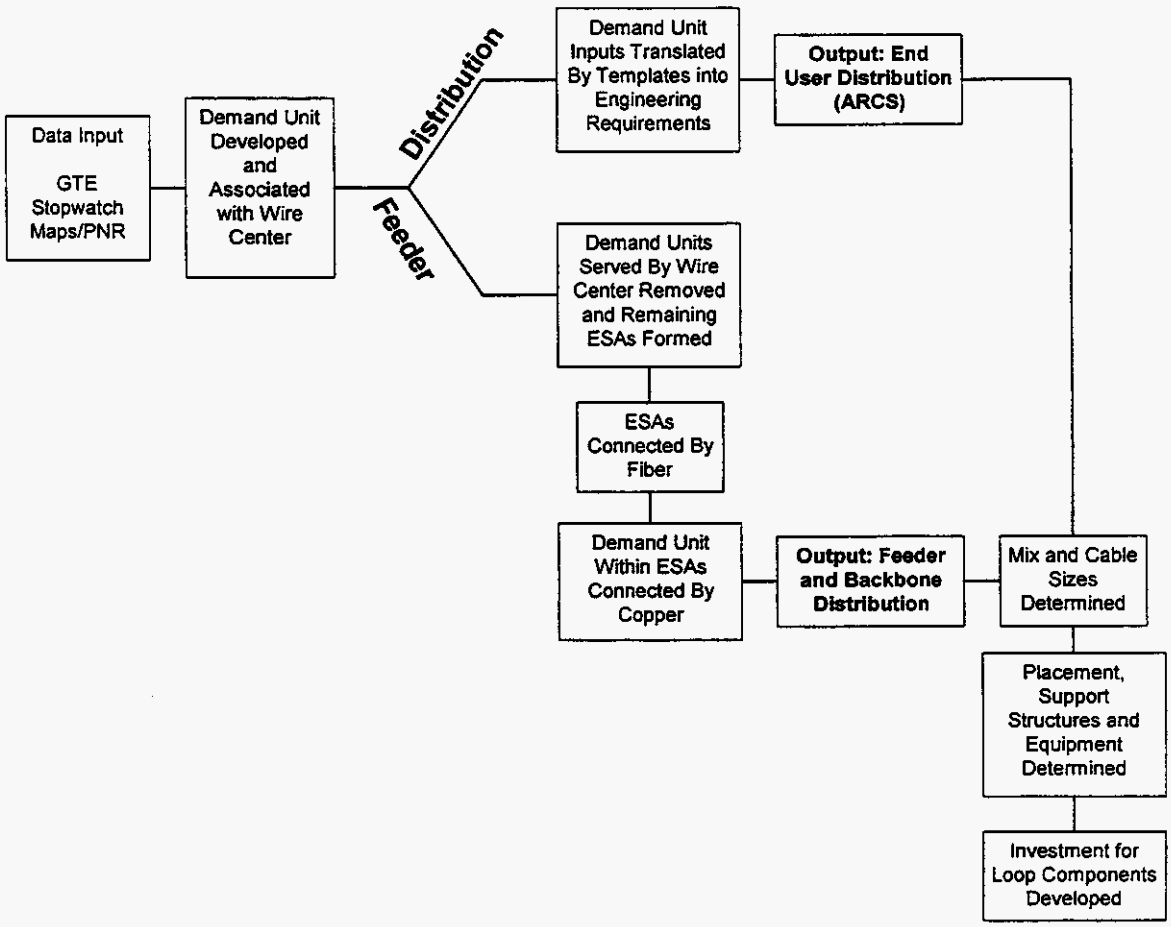


Figure 1: Loop Module Flow Chart

MODEL METHODOLOGY

Loop Module Design

The foundation of the ICM 4.1 Loop Module is based on four items. These items ensure that the Loop Module is sensitive to the demand characteristics of each area being modeled, and that the results of the module accurately estimate the cost of providing service. The four items are: uniform demand units, an Electronic Serving Area (ESA) development, a local loop network design, and detailed engineering at all levels.

The **Uniform Demand Unit** is a standard 1/200th degree by 1/200th degree area that contains detailed demand, infrastructure, and topographical information needed to accurately design a telephone network. The unit, depending on latitude, encompasses an area of 1500-by-1800-foot or about 60 acres. This size achieves precision in customer location and preserves the relation to the infrastructure that surrounds the customer.

The next item is the Loop Module's **Electronic Serving Area (ESA)** development. An ESA is an area in which all customers have access to a loop capable of providing digital services. The K-mean clustering algorithm was chosen as the mechanism for developing ESAs because of its speed, flexibility, and ease of understanding how the results were developed. Using the K-mean algorithm, line size constraints, distance constraints or topographical constraints can be easily implemented without having to modify the basic algorithm.

The third item is the Loop Module's **Local Loop Network Design.** ICM uses a Constrained Minimum Spanning Tree (CMST) Algorithm patterned after the Spanning Tree routine used by the Federal Communication Commission's (FCC's) Hybrid Cost Proxy Model (HCPM). This algorithm is used to develop the feeder network connecting the pair gain devices and the backbone network that connects the demand units within a cluster. Templates define the distribution network within a demand unit.

The final item is the **Detailed Engineering** contained in the Loop Module. ICM uses a variety of detailed engineering procedures to provide logical determinations of network characteristics such as structure type and size, placement type, material types and sizes, and labor costs.

Uniform Demand Unit

The uniform demand unit is a standard 1/200th degree by 1/200th degree area. Although the basic demand unit is not a standard size in the sense of square feet, it is standard in terms of degrees. This is a very important feature

because it precisely locates the area being examined. A demand unit contains residential and business line counts, road-feet, and topographical conditions such as a bedrock depth, water table depth, elevation, the amount of surface water, and the presence of major highways. All of the data mapped to the demand units are based on publicly available information.

The 1/200th degree demand unit is a fundamental building block of ICM. The unit is developed to represent a logical "square" of the earth's surface. The latitude and longitude of its center point or "centroid"[2] identify each demand unit. The unit, measured in latitude and longitude, and consisting of a standard size measure in degrees, is a natural way of further dividing up the coordinate system imposed on the surface of the earth. In the actual demand unit record found in the model, coordinates that indicate the position of each demand unit relative to the wire center have replaced the latitude and longitude. A value of (0,0) would indicate that the wire center is contained within the demand unit; a value of (1,0) – where X coordinate is 1 and Y coordinate is 0 – would indicate that the demand unit is immediately adjacent and to the right of the one containing the wire center.

The demand unit, besides being a standard 1/200th degree by 1/200th degree area on a map, is also a record in the Demand Table with each column of the table containing information such as CLLIs, road length, water table, bedrock etc. (See Appendix A for additional fields). When a demand unit is viewed as a record it gives a clear picture of how the information populating it is developed and what are the sources of the data.

The following sections contain a step-by-step process by which each field or column in the demand unit record is populated. The description of this process begins with the data sources of a demand unit record and finishes with a description of the process by which the record is populated.

SOURCES OF DEMAND UNIT RECORD

Road, terrain and line count information is used to develop customer location data and to reflect conditions that impact the cost of service. The road information comes from the US Census Bureau's Topologically Integrated Geographic Encoding and Reference (TIGER) files, and the terrain data comes from US Department of Agriculture's State Soil Geographic (STATSGO) database. PNR Associates provides estimates of business and residential line count data at the census block level developed from US Census data.

Stopwatch Maps compiles the line count data from PNR Associates, the road (TIGER files) data from US Census Bureau and the terrain data (STATSGO) from US Department of Agriculture, and processes the data to produce a text

[2] The centroid is labeled with latitude in the form dd.dd25 and a longitude in the form –[d]dd.dd25, where d represents a decimal digit such as the 2 and 5 in the third and fourth decimal places. Since the latitude and longitudes of the demand units will end in either 'dd0' or 'dd5', the midpoint will end with either a value of 25 or a value of 75.

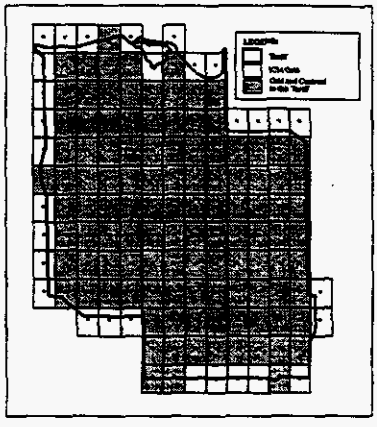
demand file (for details see Appendix B). The processing of the data involves taking the terrain and road data and distributing it along with the information provided by PNR Associates to the demand units using MapInfo[3]. This same package is used to associate the demand units with the appropriate tariff boundary and wire center. MapInfo allows the data that have been collected by Census Block or Census Block Group, (which is highly irregular in size and shape), to be accurately mapped to the standard 1/200th degree demand units.

PROCESSING OF DEMAND UNIT RECORD

Assignment Of Demand Units To Switches

GTE's tariff boundaries for each wire center are overlaid upon the demand unit network, using MapInfo, and only those demand units whose centroid falls within a GTE tariff boundary are included in the modeling process.

The example below demonstrates that although 144 demand units fall within, or otherwise intersect, this particular tariff boundary, only 114 of the demand unit centroids fall within its boundary and will be assigned to a switch within the boundary.



The demand unit assignment process begins with GTE providing Stopwatch Maps a set of tables, one containing digitized wire center boundaries and the other containing the set of switches identified by the Common Language Location Identifier (CLLI) code and Latitude and Longitude.

Development Of Coordinates

The X and Y coordinates of the wire center are (0,0). The remaining coordinates are developed relative to the wire center/central office.

[3] Map Info is a sophisticated Geographic Information System (GIS) software package.

The X coordinate is the horizontal location from the wire center/central office of a demand unit. A positive number indicates that a demand unit is that many units to the right of the central office. A negative number indicates that a demand unit is that many units to the left of the central office. A zero would indicate that it is directly above or below the central office depending on the Y value. The Y coordinate is the vertical location from the central office of a demand unit. A positive number indicates that a demand unit is that many units above the central office. A zero would indicate that it is directly to the left or right of the central office depending on the X value.

Road Length Identification Data Development Based On TIGER Files

The total road length within a demand unit is used to characterize the infrastructure development that has taken place within it and serves as the basis for the development of the distribution investments associated with the unit.

The type of roads assigned to the individual demand units consists of roads along which housing or business development would normally occur and from which the customer would have access to their premises. Excluded road types include interstate highways, limited access roads, bridges, tunnels, and access ramps. Also excluded are segments such as alleys, driveways, and motorcycle trails. Driveways and alleys are excluded because they are typically associated with premises that can be reached with one of the types of road already included. Motorcycle trails are excluded because one does not typically find residential housing or businesses located along them.

The mapping of roads to demand units involves overlaying, in MapInfo, the demand unit table onto the digitized data from the Topologically Integrated Geographic Encoding and Reference (TIGER) files and calculating the total length of the segments that fall into each of the demand units. Where a segment runs through more than one demand unit, the segment is proportionally assigned to each of the demand units. This process of assigning road length takes into account both curved and straight segments.

There are two popular formats used to identify the exact position that any given latitude and longitude occupies upon the earth. The North American Datum (developed in 1927 and known as NAD27) is the most commonly used. The NAD27 datum was used to develop the 1994 Tiger files, a product of the US Census Bureau.[4] The GTE tariff boundary and wire center data that is used as ICM input was developed with NAD27.

Assignment Of Terrain Data To Demand Unit

The *State Terrain,* a digitized census block group level data set, used by Stopwatch Maps was derived from the STATSGO data. Stopwatch Maps assigns terrain data to demand units by determining which census block group

[4] A detailed discussion of the TIGER/Line files is available on the US Census website: http:www.census.gov:80/www/tiger/content.html.

(CBG) contains a demand unit's centroid. The terrain characteristics of the CBG containing that point is assigned to the corresponding demand unit. The assignment of this data, in conjunction with the road feet data, determines the character, and, for the purposes of ICM, the *style* of a demand unit.

ICM considers two terrain elements that specifically impact the cost of providing telephone service – bedrock depth, and water table depth. Bedrock depth indicates the average minimum depth of bedrock rounded to the nearest whole number of inches. Water-table depth, similar to bedrock depth, identifies the average minimum depth of the water table in inches rounded to the nearest whole number. A more detailed description of the process is provided in Appendix B.

Line Count Development Based On Data From PNR Associates

PNR Associates, using their Access Line Model, develop the basic business and residential line counts at the Census Block level. The model uses survey information, the Dun & Bradstreet database, the Donnelley Marketing household database and Claritas 1995 census estimates, based on the 1990 Census, to estimate the number of residential and business access lines in each Census Block in the United States. Stopwatch Maps takes the information developed by PNR Associates and maps it to the demand unit using MapInfo.

The process of mapping demand quantities, which is road-based rather than area-based, begins by determining for each demand unit the road segments and portions of road segments that belong to each Census Block overlaid by the demand unit. A set of allocation factors is then developed for each Census Block, each factor corresponding to one of the demand units overlaying the Census Block. The allocation factors, which sum to one for each Census Block, are the ratio of road feet from the Census Block in the demand unit to the total road feet in the Census Block.

Once the sets of allocation factors have been developed for each of the Census Blocks they are used to assign the residential and business lines to the demand units. The resulting line counts in a cell is the sum, across all of the Census Blocks that the demand unit covers, of the allocation factors times the residential and business line counts. This allocation process produces the raw demand unit data that are then reconciled to match the total available from ARMIS reporting for the wire center. This reconciliation is done for both residential and business lines and involves scaling the line counts so that their total exactly matches that of the wire center.

Electronic Serving Area (ESA) Development

An Electronic Serving Area (ESA) is an area in which all customers have access to a loop capable of providing digital services. The size of an ESA is determined by the maximum loop length (in combination with cable gauge) that allows for the satisfactory transmission of pre-determined digital data rates and acceptable analog voice levels.

ICM 4.1 provides the user the flexibility to develop loop cost based on three types of networks which will result in two ESA sizes due to different loop lengths. The first type provides a network capable of providing 6Mbit digital transmission to all customers by developing ESAs (clusters) with maximum loops of 12,000 feet and by provisioning the loops with 24 gauge cable. The second design also develops ESAs with maximum loops of 12,000 feet but, because it designs loops with 26 gauge cable, it provides loops with limited bandwidth capability. The third network develops ESAs with maximum loops of 18,000 feet and provisions the loops with 24 gauge cable. It also utilizes DLC extended loop cards for the longer loops requiring additional gain[5]. Bandwidth capability is limited as well in this scenario due to the longer loop length.

Electronic Serving Areas are developed in ICM using a K-mean clustering routine. Clustering algorithms, which are typically used in marketing research, are commonly used for grouping observations and pattern recognition.

The K-mean methodology starts with a known number of clusters and focuses on determining the members of those clusters. The number of clusters to be used and the initial start points must be determined outside of the K-mean algorithm. Once given these values, the algorithm will cycle through a series of steps involving assignment of observations to clusters and recalculation of means (i.e. centers of clusters). During the assignment process any observation is free to shift from one cluster to another. The process will stop when all observations are associated with the cluster center closest to them.

If the number of clusters is known in advance, the K-mean methodology needs to look at only one set of clusters. This makes the methodology much more flexible because it allows detailed checks to be performed at the end of each clustering without interrupting the cluster process. To achieve this speed and flexibility advantage, the K-mean algorithm utilizes a set of driver programs that control the cluster count, develop the seed cluster, and test to determine if the stopping criteria have been met.

Local Loop Network Design

The ICM 4.1 network consists of two basic components: Feeder network and Distribution network. Each of these networks is further sub-divided into two components, as described below.

[5] Loops served by DLCs that extend greater than 16 kft from the DLC require extended loop cards.

Headings	Description
Feeder Network	Fiber Feeder
	Copper Backbone Feeder
Distribution Network	Local Distribution (Copper based)
	Copper Backbone Distribution

The ICM 4.1 feeder network is sub-divided into the fiber feeder and copper backbone feeder. The fiber feeder connects the wire center to all Digital Loop Carriers (DLCs) in an ESA. The copper backbone feeder connects the DLCs (in non-core clusters)[6] or the wire center switch (in the core cluster) to the cross connect boxes (SAIs) in each of the four directions.

The two components of ICM 4.1's Distribution network are the backbone distribution and local distribution. Both these components are copper-based. The backbone distribution connects the SAI to the local distribution portion of the network. The local distribution network completes the loop, extending from the copper backbone distribution cable to the customer. If a cross connect box is not required in the route, the copper backbone distribution connects directly to the local distribution network, eliminating the need for backbone feeder cable (refer to Figure 2 for details).

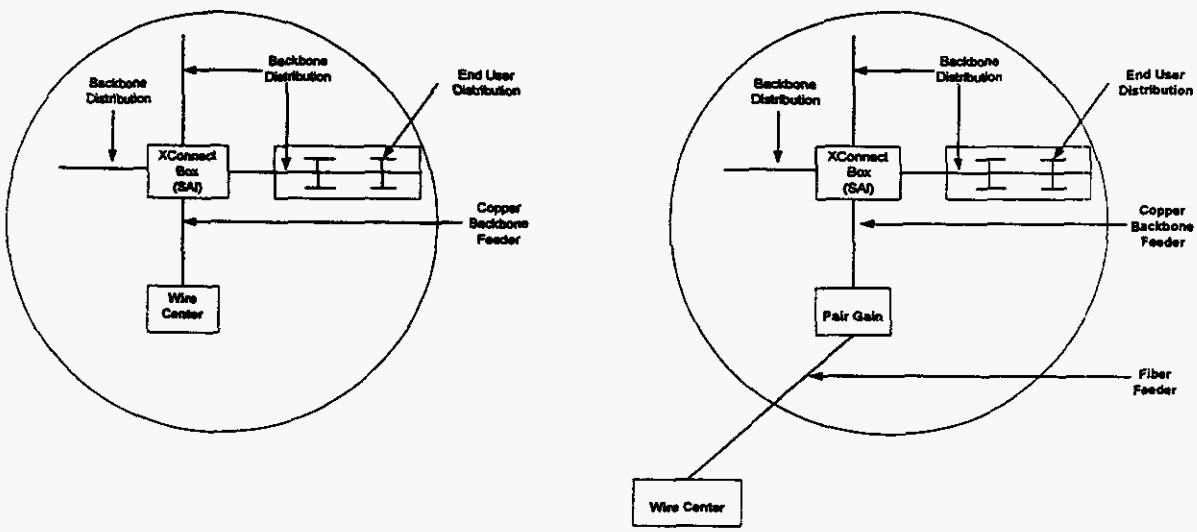


Figure 2: Core and Non-Core Local Loop Network Design

[6] The cluster that is fed on a physical basis from the central office is referred to as the "core" area. Clusters that are fed from Digital Loop Carriers (DLCs) are referred to as being in the "non-core" area.

FEEDER NETWORK AND BACKBONE DISTRIBUTION DEVELOPMENT

Both feeder components, as well as the backbone distribution cable, are designed through the use of a Constrained Minimum Spanning Tree (CMST). The CMST, like all Minimum Spanning Tree algorithms, finds a set of paths between each site so that every site is connected to the main site and the total path length is minimized. If unconstrained, the algorithm would tend to generate a network where each site has one path entering and one leaving. This tendency, when realized, produces a network that does not resemble the cable pattern typically found around a wire center. To ensure that this tendency is not realized, the constrained algorithm incorporates dummy sites called Junction Nodes. The Junction Nodes, which are pass-through sites on the x and y-axes, allow plant to be placed in each of the four cardinal directions around the wire center without violating the basic assumptions of the algorithm.

The underlying CMST algorithm used by ICM begins with a network consisting of only the wire center or ESA center, which is referred to as the Supplier Node. Additional nodes are attached to the network using a minimum distance criterion. The first step of the process involves finding the closest site or demand unit, which is referred to as a node, to the supplier node. At each subsequent step, the algorithm determines which of the nodes not yet in the network is closest to any attached node. That node is then added to the network by attaching it to the closest attached node. The algorithm proceeds in this manner until all of the nodes are in the network. The ICM CMST algorithm results in a network where the nodes are connected using right angle, or rectilinear, links along the axes.[7]

Fiber feeder and backbone cable segment lengths are calculated from the arc[8] distances between the nodes and are assigned an Arc Type based on their location in the network and their function as fiber feeder or backbone. While Distribution cable segment lengths are assigned an Arc Type, their segment lengths are not calculated in this manner but are set based on the Demand Style of the demand unit (see Distribution Template Design description in the Basic Engineering Assumptions section). Interoffice cable is also assigned an Arc Type. The development of the interoffice facility cable is discussed in the Interoffice Transport Module. The following table describes the Arc Type hierarchy used in ICM.

[7] Arcs (cable segments) may be shown on the visual interface as diagonals. However, all loop length calculations and investments utilize rectilinear distances.
[8] An arc is a cable segment.

ARCType 0	The interoffice fiber cable that connects two wire centers between the closest Digital Loop Carrier (DLC) locations in each wire center serving area.
ARCType 1	The fiber feeder cables between the wire center and DLC locations in its serving area.
ARCType 2	The copper backbone cable in the core area that connects all the demand units in the core area to the wire center.
ARCType 3	The copper backbone cable in the non-core area that connects all the demand units in the non-core cluster to the DLC site serving that cluster.
ARCType 4	The copper distribution cable within the demand unit that connects all the customers to the backbone cable.

LOCAL DISTRIBUTION NETWORK DEVELOPMENT

A distribution template is used to develop the characteristics of the local distribution plant (See Appendix C for details) and are user input. Templates allow the user to emulate some of the thought processes that go into designing a network, based on the characteristics of a demand unit. The distribution plant cost comprises a significant portion of a network's cost, and templates allow users flexibility and enable them to estimate the cost of a network with relative precision. A more detailed explanation of the distribution template is provided in the Basic Engineering Assumption section of this document.

PLANT MIX ALLOCATION

The plant mix comprises a combination of aerial, buried and/or underground fiber and copper cable. The percentages of aerial, buried and underground fiber and copper cable are inputs into ICM at the wire center level.

In the main feeder routes the underground cable closest to the wire center is placed first, followed by aerial and then buried cable. The total distances of aerial, buried, and underground fiber are determined first, then the percentages in the wire center are checked to see if 100% of one plant type is used. If not, and if no underground is present, then the arcs (cable segments) are assigned in order of decreasing demand, to be 100% aerial until the required aerial feeder cable is exhausted. Then the remaining feeder cable is assigned to buried.

Similarly, placement in the backbone and distribution is based on keeping underground cable closest to the wire center, followed by aerial and then buried cable. All core arcs are assigned first, in order of increasing arc level and decreasing demand within an arc level. After the core area arcs, the remaining backbone arcs are assigned, followed by distribution segments. In each case, the cables are assigned in order of increasing arc level and by decreasing demand within an arc level.

Detailed Engineering

Facility Investment Development

GTE MATERIAL INPUTS

Vendor contracts and GTE Advanced Materials System (GTEAMS) provide the sources of prices used in the ICM material table (see Appendix A for details). GTEAMS is the system used by GTE to perform planning, inventory accounting, and material purchasing management functions. Engineering and costing groups access GTEAMS to obtain the current base price of materials required to estimate the cost of a project or a service offering.

Current prices for all available material codes are maintained in GTEAMS. The codes include circuit equipment, support structure, and copper and fiber cable by size. Prices are kept current through regularly updated price quotes from GTE Service Corporation and GTE Supply third party vendors and invoices from current purchases to inventory.

Third party vendor price quotes represent the current prices paid by GTE Service Corporation for the selected material and used in the creation of purchase orders. These prices are updated quarterly on a mass basis, except that items having a significant (10 percent or more) price change may be updated more frequently. Prices for GTEAMS inventoried items are standard average unit prices based on a history of recent purchases from vendors. Material costs available from GTEAMS reflect economies of GTE's nationwide operations and centralized supply. However, material costs input into ICM are made state specific through the addition of state specific shipping and handling costs and minor material loadings.

Material cost and the corresponding unit of measure used in ICM are listed in the following table and are described in more detail in Appendix A.

Category	Unit
Fiber Cable	$/Foot
Copper Cable	$/Foot
Drop Wire	$/Foot
Digital Loop Carrier (DLC)	$/System & $/Line
Poles, Manhole, Pull Box	$/Unit
Conduit	$/Foot
MDF & Protectors	$/Unit
Network Interface Device (NIDs)	$/Unit
Terminals	$/Unit
Cross Connect Boxes	$/Unit

GTE LABOR INPUTS

Similarly, vendor contracts are used to develop the state-specific labor activity rates. GTE state-specific labor rates are used to develop costs for labor activities not included in the vendor contracts.

The vendor labor activity rates in the following table are weighted averages of all of the vendor contracts for a particular state. Each vendor contract specifies a rate per geographic area, or zone. In order to develop a composite rate for the state, the zone-specific rate is weighted by the percentage of the state's total road feet in that zone.

The weighting is determined as follows:

- The names of exchanges in each zone are obtained from the vendor contract.
- The CLLI code(s) associated with each exchange is determined from GTE's wire center database.
- The total number of road feet within the wire center serving area is calculated as the sum of road feet for all of the demand units assigned to that wire center.
- The total number of road feet is calculated for each zone.
- The weighting for a zone is calculated by dividing the total road feet in the zone by the total road feet in the state.

See Appendix A for detailed descriptions of these materials and labor inputs. For documentation of how state-specific inputs are developed, see the *ICM Module Support Documentation* included in this filing package.

Labor activity costs are provided for items listed in the following table:

Category	Unit
Placing Copper Cable	$/Foot
Placing Fiber Cable	$/Foot
Poles, Guys, Anchor	$/Unit
Strand	$/Foot
NIDs	$/Unit
DLCs	$/Unit & $/Line
Splicing Copper Cable	$/Pair
Splicing Fiber Cable	$/Strand
Trenching, Plowing, Backhoe, Hand Digging	$/Foot
Cutting & Removing Concrete	$/Foot
Manholes & Pull Boxes	$/Unit

Splicing Quantities

ICM allows one splice per average IPID length. An IPID (Individual Plant Identification) is the length of cable between splices in GTE's cable record system. IPID lengths are determined independently for copper and fiber cables. The average value of the IPID length is determined by the ICM user, and is entered in the user options table. A quantity of splices is then assigned by ICM to each section of cable based on cable size, IPID length, and total cable footage.

DROPS AND NID INVESTMENT

Drop Wires and Entrance Cables

To determine **residential** drop investment, each demand unit is designated as having either single family or multi-family buildings based on its number of residential units.

If the number of residential units in a demand unit is less than 500, then single family dwellings with drop wires are assumed. User input determines the size

of the drop wire (3 or 5 pair). The number of drop wires is equal to the number of residential units.

If the number of residential units is 500 or greater, then multi-family units are assumed and a 25 pair entrance cable is placed to all residences in the demand unit. The number of entrance cables is determined by assuming that each entrance cable is 50% utilized. The number of residential lines is divided by 50% of the entrance cable drop size (25/2) to determine the number of entrance cables required.

To determine **business** drop investment, the number of business units in each demand unit is considered.

If the number of business units is less than 500, then drop wires are placed. The user indicates the size of the drop wire (3 or 5 pair) in the ICM input and the number of drop wires is equal to the number of business units.

If the number of business units is between 500 and 1,250, a 25 pair entrance cable is assumed for all business locations in the demand unit. For demand units with over 1250 business units, a 50 pair entrance cable is used. The number of entrance cables is calculated in the same manner for residential or business units (50% utilization is assumed). The number of business lines is divided by 50% of the drop size (50/2) to determine the number of entrance cables required.

Average Length of Drop Wires and Entrance Cables

The average length of the drop wire or entrance cable is calculated by determining the average lot size and assuming that the drop or entrance cable originates at the corner of the lot and terminates in the center of it.

The average lot size is calculated by taking the typical area of a demand unit and dividing it by the number of residential and business units in the demand unit. Since the drop wire or entrance cable ends in the center of the average lot, it forms a diagonal across one fourth of the lot size. The length of this cable is calculated using the Pythagorean theorem as shown below:

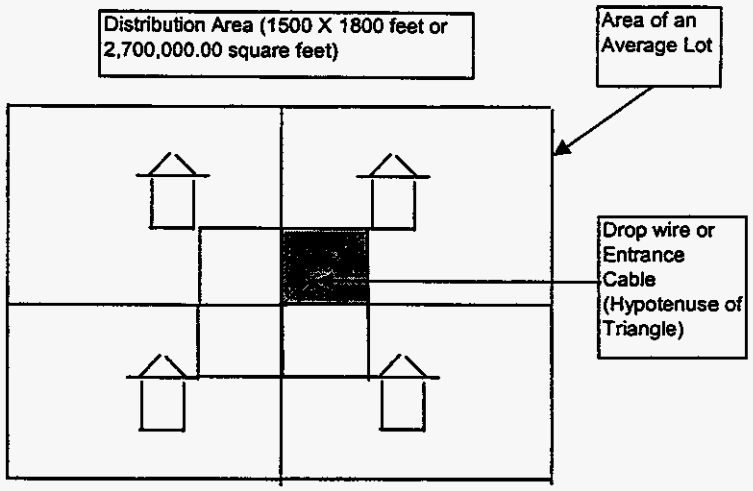


Figure 3: Calculating Drop Wire and Entrance Cable

The minimum and maximum drop lengths are user adjustable. ICM will utilize the minimum drop length in a demand unit when the calculated drop length is less than the user-input minimum length. ICM will utilize the maximum drop length in a demand unit when the calculated drop length exceeds the user-input maximum length.

Terminals and NIDs

When drop wires are used, one distribution terminal is assumed for every four residential units and for every four business units. A NID is placed for each unit.

When 12, 25 or 50 pair entrance cables are used, a 12 pair, 25 pair or 50 pair building terminal is placed. The building terminal serves as the NID. The number of building terminals is equal to the number of entrance cables in a demand unit.

Support Structure Investment Development

STRUCTURE FEET-TO-ROAD FEET RATIO

In general ICM places structure length in a wire center equal to or less than the amount of road feet in the wire center[9].

The total length of feeder and backbone arcs (ArcTypes 1 thru 3) in a wire center is divided by the total road feet in a wire center. In the majority of demand units, the result is subtracted from 1 to develop a percentage that is applied to the road feet in each demand unit to determine the amount of remaining road feet available for placement of distribution cable. This process prevents the sum of the length of feeder, backbone, and distribution from exceeding road feet length in the wire center.

There are two exceptions to this process. In very low-density demand units, ICM places no distribution cable, and in very high-density demand units, the reduction in road feet available for distribution cable placement is not made. Two user inputs on the distribution template determine the points at which these scenarios apply. The Distribution Demand Threshold (DDT), represents the demand unit demand below which no distribution cable is placed. The Arc 4 Demand Threshold (ARC4DT) represents the demand unit demand above which no downward adjustment is made in the distribution cable length.

AERIAL STRUCTURE

When placing aerial cable, ICM calculates a structure investment consisting of poles and anchor/down guy costs.

Pole spacing, a user controlled input, is used to determine the number of poles required for supporting a length of aerial cable. Total poles are calculated as follows: Poles = 1+ (total aerial cable length in section/pole spacing), with the number of poles in the () rounded to the nearest whole number. Guy wire and anchor investment is determined by the user by inputting the percentage of poles requiring anchors and down guys on the user input table.

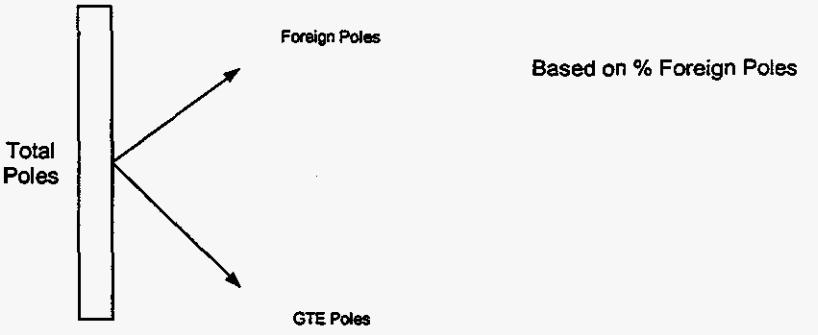
ICM assumes that 30'poles are used for the percentage of poles that GTE does not share, and uses 40' poles when sharing pole space with other utilities.

POLE SHARING

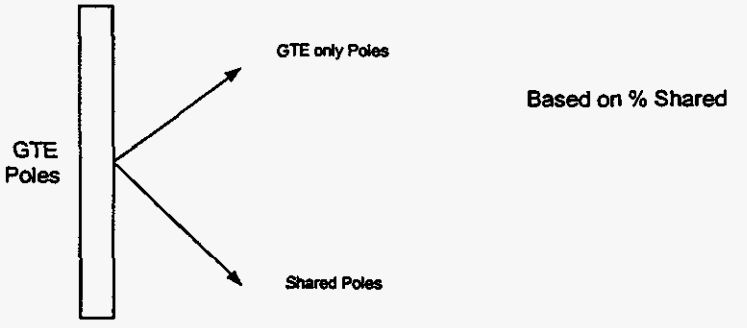
The determination of the effects of pole sharing on pole investment consists of several steps and is shown in Figure 4 below. First, the percentage of poles that are "foreign" (poles leased from power companies under contract and input by user) are subtracted from the total number of poles. Expense for leasing space on these poles from power companies is included in the cost for

[9] Certain road types are excluded from the road feet calculation. Refer to the section "Road Length Identification Data Development Based On TIGER Files" of this document for a listing of excluded roads.

structure in the Expense Module. Second, the pole sharing percentage input by the user is applied to the remaining poles to determine the number of poles that are shared by GTE. Third, GTE's portion of the shared pole investment is determined by dividing the total shared pole investment by the number of users attached, a user input. For non-shared poles, GTE is responsible for all the investment. Fourth, the total pole investment is calculated by adding GTE's portion of the shared pole investment and the investment associated with the non-shared poles.



GTE Poles = (1-% Foreign) (Total pole count)



$$\text{Investment} = \%\,\text{Sharing} * \left(\frac{\text{Shared Pole Cost}}{\#\,\text{of Pole Users}}\right) + (1 - \%\,\text{Sharing}) * \text{GTE Only Pole Cost}$$

Figure 4: Pole Sharing

BURIED STRUCTURE

For buried cable, the structure investment can include investments for plowing and/or trenching. Plowing for both distribution and feeder cable will occur if certain soil characteristics and user settings are met and certain demand levels are not exceeded. Other construction charges, including hand digging, boring, and concrete cutting and replacement do not apply when plowing is utilized.

BURIED DISTRIBUTION CABLE STRUCTURE

The Model assumes plowing in all circumstances except where more than two cables are required, where bedrock is too close to the surface to allow cost-effective plowing, or where the area is too developed to effectively plow. A user-input[10] determines whether an area is too developed for plowing.

When plowing cannot be used to place buried cable, the cable is placed with a trencher. The conditions required for plowing or trenching to occur for buried distribution cable are summarized in the table below. When a trencher is used to place distribution cable, additional labor items can be added to the user's discretion to reflect increased route congestion and urban conditions. These items[11] include boring, hand digging, concrete removal and replacement. Boring, hand digging and concrete removal and replacement activities are primarily found in urban areas. Rock sawing is needed when the bedrock is closer to the surface, and can occur in rural as well as urban areas. The Model determines the rock-sawing requirements.

Condition	Distribution - Plow	Distribution - Trench
Plow Flag	1 = Plowing Allowed	0 = No Plowing
Depth to Bedrock	> 30"	< 30"
Sharing (Users- Trench under Options/User)	< = 2 Users	>2 Users
Separation	Random	Random

BURIED FEEDER AND BACKBONE CABLE STRUCTURE

As in the distribution network, feeder and backbone cables will be plowed when three conditions are met. The first condition, analogous to the '0' and '1' setting of the PLWFLG indicator in the distribution network, involves the density of the wire center. Feeder and backbone cables may be plowed only when the wire center serving area is labeled as a low density wire center (less than 50 lines per square mile - designated as Low Density in the XXNODES.DB table).

[10] The Plow Flag (PLWFLG) in the Distribution Style Table (see Appendix A) must be set to a value of 1 for the demand unit style in which plowing is desired. The PLWFLG default settings on the Distribution Style Table allow for plowing in the three demand unit styles that contain the lest road feet. The default settings reflect the fact that plowing is most practical in areas with little road feet, i.e., rural, open areas with few underground obstructions.

[11] These items are user-input and controlled by the Structure Flag (STRFLG) on the Distribution Style Table (see Appendix A). A '0' in the STRFLG field indicates that no additional items are required. A '1' in the field will add the additional items on a percentage basis as input in the user inputs. And, a '2' in the field reflects a very dense scenario with many underground obstacles.

The second condition states that only non-shared construction may be plowed. Since random sharing is not allowed in the feeder network, the required separation between GTE and other facilities cannot be achieved by plowing the cable.

Third, the bedrock must be below the surface far enough to allow sufficient cover, i.e., 30" for copper cable and 48" for fiber cable. However, to avoid the additional expense of trenching and rock sawing for fiber placement when bedrock is between 30" and 48", ICM allows fiber cable to be plowed at 30" within a protective subduct.

Additionally, the cost of pre-ripping, an activity used in conjunction with cable plowing to loosen up very hard soil conditions, is added to the cost of plowing fiber cable in ICM at a 10% rate. For example, if 1000' of fiber cable is placed, 100' is assumed to require pre-ripping. Pre-ripping is not applicable to copper cable placement.

As in distribution cable a trencher is used to place feeder or backbone cables when plowing is not possible. Also, as with distribution cable, the user options for the percent boring, hand digging and concrete removal and replacement are applied only under certain conditions. Unlike distribution cable, however, the wire center density is the sole determinant of the application of these factors. If the wire center is high or medium density (over 50 lines per square mile) the user input percentages of boring, hand digging, and concrete removal and replacement are applied. No boring, hand digging, or concrete removal and replacement are applied in low-density wire centers.

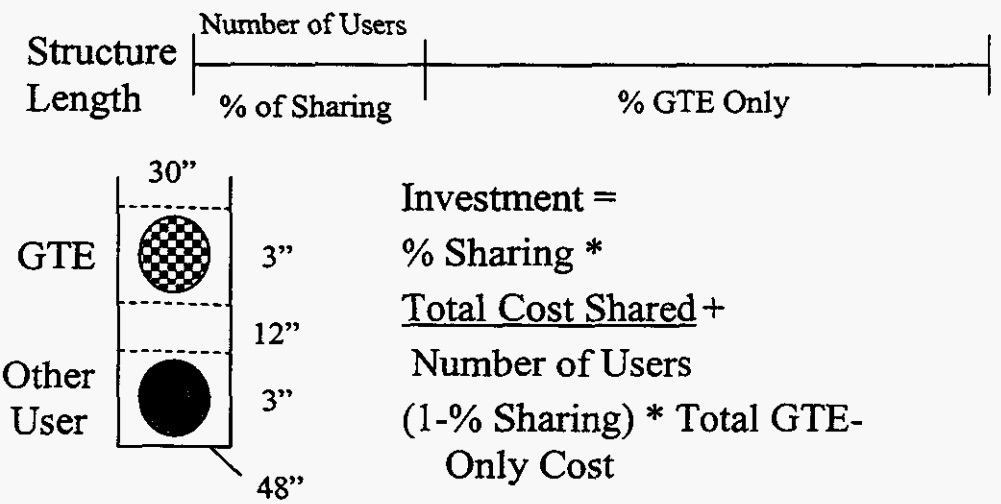
In non-sharing scenarios for feeder and backbone cables, rock sawing is added to the cost of trenching if bedrock is within 30" of the surface. If sharing exists, rock sawing is added to the cost for the trench if bedrock is within 42" of the surface to allow for the placement of the sharing facility. See figure 4 for a pictorial representation of the sharing scenario in feeder and backbone facilities.

An overview of the conditions required for feeder and backbone cables to be plowed or trenched is provided below:

Condition	Feeder - Plow	Feeder - Trench
Demand in Wire Center (as shown in Density column of XXNODES.DB)	< 50 lines per square mile (Low Density)	> 50 lines per square mile (Medium or High Density)
Fiber Feeder - Depth to Bedrock (Soil in XXNODES.DB)	> 48" no subduct required 30"- 48" subduct required	< 30"
Copper backbone - Depth to Bedrock (Soil in XXNODES.DB)	> 30"	< 30"
Sharing	Not permitted - 12" separation required	Sharing permitted

TRENCH SHARING

Trench sharing is shown in Figure 5 below. As with conduit and pole sharing, a user input determines the extent of trench sharing. For shared trenches, the total cost of the trench is divided by the number of parties who use the trench (including GTE) to determine the portion of the shared investment for any party, including GTE. For non-shared trenches, GTE is responsible for all of the investment. The shared and non-shared trench investments are weighted using the percentage trench-sharing factor.



User Input: total number of utilities using the trench including GTE

Figure 5: Trench Sharing

UNDERGROUND STRUCTURE

Underground structure consists of ducts, sub-ducts, manholes, and pullboxes. Underground cable is always placed inside ducts. ICM places a minimum of two ducts in underground facilities, with additional ducts and pullboxes and/or manholes placed when the situation warrants.

UNDERGROUND DISTRIBUTION CABLE STRUCTURE

In the distribution network, the placement of pullboxes and manholes depends on the business line demand in the demand unit. In demand units with six or less business lines, ducts are placed without pullboxes or manholes. In demand units containing between 7 and 60 business lines, pullboxes are placed as long as the number of ducts required does not exceed two. The total number of ducts is the sum of ducts used by GTE and other users and is determined by cable demand and the number of duct users shown in the inputs. When demand units contain more than 60 business lines, manholes are placed. A user input determines manhole and pullbox spacing for distribution cable.

UNDERGROUND FEEDER AND BACKBONE CABLE STRUCTURE

For backbone cable (copper), pullboxes are used if the demand used for cable sizing is less than 400 lines and two or fewer ducts are required. As in the distribution network, the total number of ducts is the sum of ducts used by GTE and other users and is determined by cable demand and the number of duct users shown in the inputs. Manholes are placed when either the demand exceeds 400 lines or more than two ducts are required. Also as in the distribution network, a user input determines manhole and pullbox spacing for backbone cable.

In the fiber feeder network, fiber cables are placed in 1" subducts inside 4" ducts. Up to three 1" subducts can be placed in the 4"ducts. In the vast majority of cases, pullboxes are placed in the fiber feeder network and are spaced according to the user established pullbox spacing. However, in the rare instance that more than six fiber cables are required along a route, manholes will be placed along the fiber route and spaced according to the pullbox spacing input.

DUCT PLACEMENT

ICM places a minimum of two ducts for underground facilities using a trencher to provide a 30" cover. If the cable demand and sharing require that more than two ducts be placed, a backhoe is used. The initial depth setting for a backhoe is for a 36" deep trench, which would provide the necessary 30" cover for 2 ducts. However, when 3-12 ducts are required, ICM applies the required additional 12" of trench depth, and if more than 12 ducts are placed, a second additional 12" of trench depth is added. The appropriate rock sawing investment is applied to the trenching if bedrock is within 36" of the surface.

In addition, investment for concrete removal and replacement is applied per the percentages input by the user.

Regarding the type and size of duct system, the standard duct formation sizes are 2, 4, 6, 9, 12, 15, 18 and 21. The number of 4" ducts required for copper cable placement equals the number of cables required that fit the standard formation. For example, if two copper cables are required, two ducts are placed. If three copper cables are required, four ducts are placed. The number of 4" ducts required for fiber cable equals the number of fiber cables divided by 3 (rounded up) that fit the standard formation.

MANHOLE PLACEMENT

As mentioned, extensive underground cable construction, when defined in ICM by certain demand parameters, requires placement of manholes. Manholes are generally placed using a bid procedure. ICM uses a GTE broad-gauge price to estimate the cost of manhole placement. In addition, ICM recognizes the effect that bedrock and the water table has on manhole placement. If the water table level is within 48" of the surface, well points are added. Likewise, if bedrock is located within 72" of the surface, ICM adds additional labor to place the manhole.

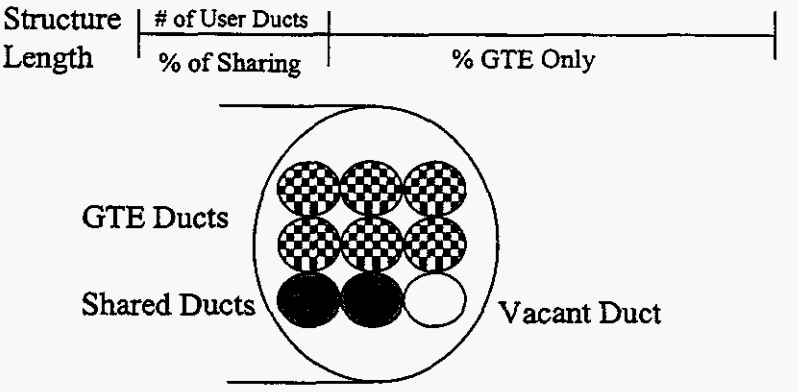
PULL BOX PLACEMENT

Pull box placement rates have been established in GTE's standard construction contracts and are used in ICM to reflect the cost of pullbox placement. The placement cost of a pullbox is increased when bedrock is within 36" of the surface. Unlike manhole placement, however, pullbox placement is not affected by the presence of a high water table.

CONDUIT SYSTEM SHARING

Conduit sharing is shown in Figure 6 below. The number of ducts is based on the total ducts needed by all parties (GTE and others). In the sections of the conduit that are shared between GTE and others, the number of ducts required by the other parties is added to the number of ducts required by GTE. The Loop Module determines the number of ducts required by GTE based on the number of working lines. The number of ducts required by other parties is also a user input. The total shared conduit investment is based on the total duct requirement of all parties. GTE's portion of the total shared investment is calculated by dividing the number of GTE ducts, including required vacant ducts, by the total number of ducts.

In sections of the conduit that are not shared with other parties, the conduit is sized based on the requirement of GTE only. GTE is responsible for the entire investment of non-shared conduit. The percentage of sharing of conduit is also a user input.



Investment = % Shared * Total Cost of Shared System * (# GTE & Vacant Ducts/Total Ducts) + (1 - % Shared) * Total Cost of GTE Only System

User Input: Ducts Required for Other Companies' Use

Figure 6: Conduit Sharing

Distribution Template Design

ICM utilizes a set of nine distribution templates to emulate distribution network designs based on varying ranges of road feet in a demand unit (see Table below). The templates were designed by starting with a demand unit that contained an amount of road feet that fit into the highest range. This demand unit contained a crosshatch pattern of streets (See the patterns in Appendix C). The nine distribution styles were then developed in steps (pruning cable sections from the previous styles) to ultimately place the minimum number of cable sections required to account for the minimum range of road feet.

The specific designs were used to reflect the cable patterns that would typically be placed as the road feet increases in a demand unit from very little road feet to the maximum road feet. The styles start with two single sections splitting off the backbone and expand to four cable networks containing several sections each. Each design section represents a section of distribution cable that would be required to serve the road feet in the demand unit. Each design segment is a collection of sections with the same demand.

The entire density range, from the least dense area to the most dense areas are modeled by these designs. In rural areas (little road feet in the demand unit) the design models only a few short cable sections splitting off from the backbone cable. This design reflects just a few side streets being served from the main road where the backbone would be placed. As the road feet increase, the number of cable sections increases to reflect a more complex street layout. The distribution style #9 models a very dense street pattern.

Distribution Styles	Minimum Value of Road Feet	Maximum Value of Road Feet
1	0	500
2	501	1000
3	1001	3000
4	3001	5000
5	5001	7000
6	7001	9000
7	9001	11000
8	11001	12500
9	12501	99999

In the above table, Column One lists the nine distribution styles, and Columns Two and Three list the minimum and maximum value of road feet that a given distribution style will support. The selection by a user of a particular style, out of the nine distribution styles listed in the above table, will depend on the ranges of road feet in demand unit.

The Distribution Style Table (Appendix A) contains the coefficients that mathematically represent the length and size of the sections developed through the nine distribution templates. In addition it also contains a set of inputs that allow users to perform sensitivity analyses on the distribution network. These inputs include the plow flag (PLWFLG), the structure flag (STRFLG), the structure high-density threshold (StrHDT), the average length fraction (AvgLenFrac), the distribution demand threshold (DDT), and the ARC4 demand threshold (ARC4DT).

For a description of all items included in the Distribution Style Table, reference Appendix A, Distribution Style Table. Also, see the 'Buried Structure' section earlier in this document for a discussion on the PLWFLG and STRFLG settings.

Cross Connect (SAI) Development

ICM 4.1 utilizes user-controlled demand and distance parameters to position and size Serving Area Interface (SAIs) in the network. Following is a description of the method GTE uses to locate and size SAIs in ICM 4.1.

DEFINITIONS:
Core Clusters: Cluster that surrounds the wire center. All points in the core cluster are within a pre-determined distance from the wire center, usually 12kft.

Non-Core clusters: Any cluster containing points beyond the 12kft pre-determined distance from the wire center.

Primary SAI: In non-core clusters this SAI is placed adjacent to the DLC at the center of the cluster. Primary SAIs are not placed in core clusters.

Secondary SAI: In core and non-core clusters, this SAI(s) is placed along a route(s) away from the center of the cluster.

Critical distance: User input, which establishes the minimum distance from the center of the cluster (wire center in core clusters, DLC in non-core cluster) to a possible secondary SAI location. A secondary SAI will not be placed closer to the cluster center than this distance. This input is listed under the OSP feeder options and is labeled 'Min Distance'.

Primary demand trigger: User input demand level that triggers placement of a primary SAI in a non-core cluster. Not used for core clusters. This input is listed under the OSP feeder options and is labeled 'Min Size 1'.

Secondary demand trigger: User input demand level that triggers placement of a secondary SAI. Used in core and non-core clusters. This input is listed under the OSP feeder options and is labeled 'Min Size 2'.

Secondary size factor: User input factor, which is multiplied by route demand to calculate the secondary SAI sizes. Default is 3.0. Used in both core and non-core clusters. This input is listed under the OSP feeder options and is labeled 'Factor'.

SAI LOCATION AND SIZE DETERMINATION

The determination of SAI size and location is dependent on the type of cluster (core or non-core) and the four user inputs defined above: Critical distance, primary/secondary demand triggers, and secondary size factors.

In both types of clusters, demand is accumulated on a route basis to determine if a secondary SAI is required on the route and, if so, the SAI size(s). Demand is accumulated from the end of the route toward the cluster center until it equals the demand required to place the maximum size SAI (maxdemand) or until the critical distance is encountered.

If maxdemand is reached before the critical distance, the maximum size SAI is placed and the demand accumulation begins again, taking the excess demand from the original accumulation into consideration. For example, if the first demand accumulation results in 100 lines more than required to place the maximum size SAI, the 100 lines are included in the second demand accumulation. This process continues until the critical distance from the center is encountered. The demand remaining at the critical distance is then multiplied by the secondary sizing factor to determine the proper size SAI at the critical distance point. The secondary sizing factor is defaulted to 3.0, which signifies the typical SAI setup of one pair 'in' and two pairs 'out'. If the route critical distance is encountered before maxdemand is accumulated, the demand at the critical distance point is multiplied by the secondary sizing

factor to determine the SAI size at the critical distance point. Finally, if the demand accumulated for the route at the critical distance point does not exceed the secondary demand trigger, a secondary SAI is not placed along that route.

CORE CLUSTER SAIS

From an SAI perspective, core clusters differ from non-core clusters in one aspect. Whereas non-core clusters contain primary and secondary SAIs, core clusters contain only secondary SAIs, i.e., SAIs placed along a route, and do not contain primary SAIs. This recognizes the fact that the MDF of the switch can generally serve as a surrogate SAI. Therefore, primary SAI algorithms and user inputs do not apply to core area clusters.

NON-CORE CLUSTER SAIS

Secondary SAIs in non-core clusters are located and sized in the same manner as core cluster SAIs. In addition, non-core clusters can also contain primary SAIs. Non–core clusters are served by Digital Loop Carriers (DLCs) with a built-in cross-connect strip that is generally large enough to cross connect approximately 200 pairs. When the capacity of the internal strip is exceeded, it becomes necessary to place a primary SAI, i.e., an SAI at the center of the cluster. The primary SAI differs functionally from the secondary, or route, SAIs in that the primary SAI does not specifically provide a feeder/distribution interface, but provides an access point for technicians to rearrange and redirect circuits without having to gain access to the DLC.

The primary SAI is placed if the cluster demand exceeds the primary demand trigger set by the user, typically 200 lines. Because the primary SAI is not used as a feeder/distribution interface, it is sized on the basis of 2 times demand, which sizes a SAI large enough to provide for one pair 'in' and one pair 'out'.

Engineering Factors

Engineering factors are user adjustable inputs that are applied to the existing demand to correctly size cables for current and anticipated demand according to GTE practices. The Distribution Engineering Factor is applied to demand in copper distribution cable and the Feeder Engineering Factor is applied to demand in copper feeder cables.

Average Loop Length Calculation

The average loop length is calculated based on the feeder and backbone loop lengths to the demand point plus the average distribution length inside the demand point (average ARC4). The average distribution length inside the demand point is calculated by applying the average length fraction (AvgLen Frac) in the distribution style table (see Appendix A) to the adjusted road footage in the demand point. The average length fraction of a particular

distribution demand style is the pre-determined percentage of adjusted road footage in the demand point that constitutes the average distribution length inside the demand point.

Loop Module Outputs

Below are the key investment outputs from the Loop Module. The total number of lines and the investment by network component are passed to the Mapping/Report Module, where monthly costs are developed. The number of lines per wire center is passed to the Switch Module for sizing purposes.

Loop Module outputs include investments for:

- Feeder Cable
- Distribution Cable
- Support Structures (poles, conduit, manholes, pullboxes, etc.)
- Terminals
- Drops
- NIDs
- NGDLCs
- Main Distribution Frames (MDF)
- MDF Protectors.

CONCLUSION

ICM 4.1 provides the most accurate estimates of the long run incremental costs of the local loop. The distribution areas are defined by 1/200th degree by 1/200th degree demand units that provide a 4-fold increase in resolution over ICM's previous demand units. An efficient and accurate clustering algorithm creates groups of these demand units in order to define pair gain locations or ESAs. The pair gain locations are connected to the wire center using a constrained minimum distance spanning tree (CMST) algorithm. This minimum spanning tree method is also used to model the backbone loop between the pair gain locations and the serving area interfaces. Distribution plant layout is modeled by the use of road feet and typical street layouts, or "demand style templates."

Loop cost outputs from ICM reflect forward-looking technologies, vendor prices, and provisioning rules. For example, the outside plant placement is based on having underground cable closest to the wire center, followed by aerial, and then buried cable. All of the most current cable, electronics, structure, and labor prices are used. The model also accounts for both the sharing of cable between feeder and interoffice routes and the sharing of structure between GTE and other carriers.

APPENDIX A: Data Input Tables

The data input tables are listed below. The tables can be viewed and printed under the menu option **Tables/Loops.** The name of each database file is listed; often the first two letters indicate state (that is, in "xxexbase.db", "xx" would correspond to the state). A user may modify data in all of the tables listed below. Detailed descriptions of each of the tables are provided in the following section.

Following the description of data input tables, a bubble chart for Distribution and Feeder components of the Loop is provided under the heading 'Processing Flow'. The Distribution and Feeder bubble charts contain a graphic presentation of the database files that are inputs to and outputs of these two components.

Loops

- Demand
  - Demand Data (XXDEMAND.DB)
  - Distribution Style (XXTEMPLT.DB)
- Material
  - Outside Plant Material (XXMATL.DB)
- Placement Costs
  - Outside Plant Placement Costs (XXLABR.DB)
- Processing Flow
  - Distribution
  - Feeder

Demand Table [XXDEMAND.DB]

Purpose: Lists geographic and demographic characteristics (road length, bedrock, water table etc.) of each demand unit.

For More Information: See Loop Module - Commercially Available Data and Appendix B – Stopwatch Maps

Note: This table is very large and printing it is not recommended.

COLUMN HEADING	DESCRIPTION
CLLI	The 11-character CLLI code of the switch within the wire center's boundaries that is closest to the centroid of the demand unit
X	The horizontal location from the central office of a demand unit. A positive number indicates that it is that many units to the right of the central office. A negative number indicates that it is that many units to the left of the central office. A zero would indicate that it is directly above or below the central office depending on the Y value.
Y	The vertical location from the central office of a demand unit. A positive number indicates that it is that many units above the central office. A negative number indicates that it is that many units below the central office. A zero would indicate that it is directly to the left or right of the central office depending on the X value.
MAP INFO	GTE generated character string used to identify the area that a switch serves.
ROAD LENGTH	Total length of road, in feet, within a demand unit
BEDROCK	Average minimum depth to bedrock in inches. The measure will vary from region to region.
WATER TABLE	Average minimum depth to water table in inches.
RES LINES	Number of residential telephone lines is adjusted to reflect GTE's ARMIS line counts per wire center.
BUS LINES	Number of business telephone lines adjusted to reflect GTE's ARMIS line counts per wire center.
WATER FLAG	True/False options (the flag is set but is not currently in use).
ELEVATION CODE	Elevation within the demand point in feet. Measures level above ground and the unit of measure is 1000 feet (Not currently used).
HIGHWAY FLAG	True/False options (the flag is set but is not currently in use).
X CLUSTER	Identifies cluster location relative to Y-axis.
Y CLUSTER	Identifies cluster location relative to X-axis.

Distribution Style Table [XXTEMPLT.DB]

Purpose: Specifies the nine distribution styles by number of road feet. Distribution styles are primarily identified by the number of road feet contained within the demand unit boundary. A distribution style was developed to represent each category of road feet and to serve as the starting point for the unique distribution architecture developed for each demand unit.

COLUMN HEADINGS	DESCRIPTION
Minval	Minimum number of road feet value for distribution style
Maxval	Maximum number of road feet value for distribution style
RCOEF1	Coefficient used to determine amount of cable feet associated with a particular cable segment within the minimum/maximum road feet value.
RCOEF2	Coefficient used to determine amount of cable feet associated with a particular cable segment within the minimum/maximum road feet value.
RCOEF3	Coefficient used to determine amount of cable feet associated with a particular cable segment within the minimum/maximum road feet value.
RCOEF4	Coefficient used to determine amount of cable feet associated with a particular cable segment within the minimum/maximum road feet value.
RCOEF5	Coefficient used to determine amount of cable feet associated with a particular cable segment within the minimum/maximum road feet value.
RCOEF6	Coefficient used to determine amount of cable feet associated with a particular cable segment within the minimum/maximum road feet value.
RCOEF7	Coefficient used to determine amount of cable feet associated with a particular cable segment within the minimum/maximum road feet value.
RCOEF8	Coefficient used to determine amount of cable feet associated with a particular cable segment within the minimum/maximum road feet value.
DCOEF1	Determine the demand associated with a particular cable segment
DCOEF2	Determine the demand associated with a particular cable segment
DCOEF3	Determine the demand associated with a particular cable segment
DCOEF4	Determine the demand associated with a particular cable segment
DCOEF5	Determine the demand associated with a particular cable segment
DCOEF6	Determine the demand associated with a particular cable segment

COLUMN HEADINGS	DESCRIPTION
DCOEF7	Determine the demand associated with a particular cable segment
DCOEF8	Determine the demand associated with a particular cable segment
PLWFLG	Plow Flag 0 = can not plow; 1 = can plow if soil conditions permit
STRFLG	Structure Flag 0 = Does not Allow Percent Hand Dig, Percent Bored, and Percent Concrete to be applied, 1 = Uses User Input Percent Hand Dig, Percent Bored, and Percent Concrete, 2 = Uses preset Percent Bore, Percent Concrete values
STRHDT	Structure High Density Threshold. Demand unit business unit level above which the preset percent bore and percent concrete placement percentages (STRFLG=2) are utilized. Will override a STRFLG setting of 1.
AVGLENFRAC	Derives the weighted average distribution loop length
DDT	Distribution Demand Threshold. When demand unit demand is less than DDT no Arc 4 distribution cable is placed.
ARC4DT	Demand threshold per sq. mile for Arc type 4 only. If square mile demand is >= to the ARC4DT value the road-feed adjustment is set to 1

Outside Plant Material [XXMATL.DB]

Purpose: Lists the material unit prices for outside plant hardware. This table is used in both the Loop and Transport Modules. Each field in the OSP Material Table is defined below.

COLUMN HEADINGS	DESCRIPTION
MATERIAL TYPE	Abbreviation for the type of equipment (for example, CUBUR for copper buried cable and POLE30 for a 30-foot pole). If the equipment is made up of multiple components, they are listed accordingly in the OSP Material table below.
SIZE	Size of the outside plant material (for example, 600 for a 600-pair underground copper cable).
DESCRIPTION	Brief description of the equipment and its purpose.
UNIT PRICE	Investment in the equipment per unit. This cost includes all components associated with any defined material type.
MINOR MATERIAL	Indicates Major or Minor Material Item (True or False). If True, the item is classified as Minor and its cost is not used towards network investment. If False, the item is classified as Major and its cost is used towards network investment.

OSP Material Input Descriptions

Below is a table describing the outside plant material inputs used in developing the investments for loop and transport components. The user can modify the prices associated with these materials and/or insert or delete inputs. Adding a material input may require the addition of a related labor activity input.

• Labor code conventions for Digital Loop Carrier (DLC) are as follows:

1st & 2nd Characters		3rd Character		4th Character		5th Character	
LC	Loop Carrier	B	Basic	A	Getting Started	R	Retail Price
		C	Coin	B	Per Line	W	Wholesale Price
		D	Digital Data Service	C	Loop Extender		
		I	Integrated Services Digital Network (ISDN)				
		4	4 Wire				

MATERIAL TYPE CODE	MATERIAL TYPE	SIZES	DESCRIPTION	UNIT
ADM	Add Drop Multiplexer Equipment	3 DS-3 12 DS-3 48 DS-3	Used with OC-3, OC-12 and OC-48 SONET rings and converts between optical signals and electrical ones. The electrical signals can be at the DS-1 or DS-3 level. ADMs allow traffic from other nodes terminating at this node to be pulled off the ring. ADMs also allow traffic originating at this node to be added to the ring for distribution to other nodes. Traffic transiting the node is converted from optical to electrical, "cleaned up" (retimed, regenerated, etc.) and converted back to optical. Includes material loading.	Per ADM
ANCHOR	Anchor	1	10" Anchor/Guy/Guy Guard: The hardware used to balance the strain on poles caused by cable and/or wire tension and to hold or counter unbalanced stress that usually occurs at corner and dead-end poles. Includes associated engineering costs and material loading with the exception of costs associated with minor materials.	Per pole
CB	Channel Bank Common Equipment	1	Multiplexes 24 voice grade and/or data circuits into a DS-1. This channel bank contains two 24 channel shelves (digroups) for a total of 48 circuits that are multiplexed onto two DS-1 circuits. Includes material loading.	Unit
CBCARD	Channel Bank Card	1	A device used to interface a dedicated special service loop coming in to the phone company to a channel bank. Includes material loading. The channel bank Channelizes 48 DS0 circuits onto two DS-1 circuits.	Per Card

MATERIAL TYPE CODE	MATERIAL TYPE	SIZES		DESCRIPTION	UNIT
CONCRETE	Concrete	1		Ready mixed concrete for replacing concrete removed in the trenching process.	Linear foot
COND	Conduit	1 2 4 6 9 12 15	18 21 24 27 30 33 36	4" PVC pipes that are placed underground and used to pass telephone cables through. Includes associated engineering costs and material loading.	Linear foot
CONNCOAX		1		N/A Input code no longer applicable	
CONNFIBR		1		N/A Input code no longer applicable	
CONNTWPR		1		N/A Input code no longer applicable	
CUAER24	24 Gauge Copper Pair Aerial	25 50 100 200	300 400 600 900	24 gauge copper cables strung outside on telephone poles. Includes the copper pairs encased in protective sheathing. Includes associated engineering costs and material loading.	Sheath foot
CUAER26	26 Gauge Copper Pair Aerial	25 50 100 200	300 400 600 900	26 gauge copper cables strung outside on telephone poles. Includes the copper pairs encased in protective sheathing. Includes associated engineering costs and material loading.	Sheath foot
CUBUR24	24 Gauge Copper Pair Buried	25 50 100 200 300 400	600 900	24 gauge copper cables plowed, bored, or placed in a trench and then covered. Cables are protected against water and sharp rock damage with protective sheathing. Includes associated engineering costs and material loading.	Sheath foot
CUBUR26	26 Gauge Copper Pair Buried	25 50 100 200 300 400	600 900	26 gauge copper cables plowed, bored, placed in a trench and then covered. Cables are protected against water and sharp rock damage with protective sheathing. Includes associated engineering costs and material loading.	Sheath foot

MATERIAL TYPE CODE	MATERIAL TYPE	SIZES		DESCRIPTION	UNIT
CUUND24	24 Gauge Copper Pair Underground	25 50 100 200 300 400	600 900 1200 1500 1800 2100	24 gauge copper cables pulled through conduit. Includes associated engineering costs and material loading.	Sheath foot
CUUND26	26 Gauge Copper Pair Underground	25 50 100 200 300 400 600 900	1200 1500 1800 2100 2400 2700 3000	26 gauge copper cables pulled through conduit. Includes associated engineering costs and material loading.	Sheath foot
DCSC	Digital Cross-Connect System	DS-1/DS-0 DS-3/DS-1		The equipment that multiplexes and demultiplexes electronic signals and acts as a means to electronically cross-connect facilities. These are sometimes referred to as Digital Access and Cross-Connect Systems (DACS). The types of digital cross connect that are included are DCS 1/0 (between DS-1 and DS-0) and DCS 3/1 (between DS-3 and DS-1). The DCS 1/0 is not utilized for multiplexing or demultiplexing but strictly for cross-connecting facilities. The channel bank (CB) is utilized for multiplexing and demultiplexing DS-1/DS-0 signals. Includes material loading.	Per DACS
DCSP	Digital Cross-Connect System Port Equipment	DS-1/DS-1 DS-3/DS-1 DS-3/DS-3		The point of access into the Digital Cross-Connect System Network (DACS). DCS Port Equipment specifically defines interfacing line capacity of the DCS. This is a variable sensitive unit. If the DACS is used for multiplexing, this equipment performs the multiplexing. Includes material loading.	Per DACS
DS	Digital Signal Port Card	DS-1 DS-3		The ADM plug-in card investment required for interfacing DS1 and DS3 signals. The ADM plug-in cards convert the optical signal into asynchronous DS1 (using the DS1 Port Card) or DS3 (using the DS3 Port Card) signals. Includes material loading.	Per Card
DSIEJ	Digital Signal Interconnect Electronic Jumper	DS-0 DS-1 DS-3		A wire that serves to connect the outside plant network to the switch within the wire center. Includes material loading.	Per pair
DSIET	Digital Signal Interconnect Tie Cable Jumper	DS-0 DS-1 DS-3		Similar to DSIEJ. A wire used to connect cable between two distribution frames or DSX panels within the wire center. Includes material loading.	Per pair
DSIOJ		DS-3		N/A Input code no longer applicable.	

MATERIAL TYPE CODE	MATERIAL TYPE	SIZES	DESCRIPTION	UNIT
DSITB	Digital Signal Interconnect Term Block	DS-0 DS-1 DS-3	This serves as the interconnect point of the outside plant to the inside plant equipment. The inside plant and outside plant cable are tied to the Interconnection Terminal Block to create an electrical connection. Includes material loading.	Per block
DSXC	Digital Signal Cross-Connect Common Equipment	DS-1 DS-3	Digital Signal Cross-Connect Level 1 and 3. These devices allow for the cross connecting of DS-1 and DS-3 lines. Includes material loading.	Per cross-connect
DSXJ	Digital Signal Cross Connect Jumpers	DS-1 DS-3	The wire or cable that cross connects DSX panels at the DS-1 or DS-3 signal rate. Includes material loading.	Per pair
DWIRE	Drop Wire	3 Pair 5 Pair 25 Pair 50 Pair	A copper service wire that is the loop component used to transport service from the distribution terminal to the customer's NID. Three and five pair drop wire costs include the material loading with the exception of costs associated with minor materials. 25 and 50 pair drop wire costs include those associated with engineering and material loading.	Sheath foot
FDPC	Fiber Distribution Panel Common Equipment	1	The Fiber Distribution Center accepts up to twelve Connector Panel Modules (6 connections/module). Includes material loading.	Per FDC
FDPM	Fiber Distribution Panel Module	1	This module connector serves as the interface between the ADM and the outside plant facilities. Here, fibers from the outside plant environment are terminated on the panel and connected through to the inside ADM equipment fiber patch cords. Includes material loading.	Per module
FIAER	Fiber Strand Aerial	6 fiber 12 " 24 "	Fiber cable enclosed in protective sheathing that is strung on telephone poles. Includes associated engineering costs and material loading.	Sheath foot
FIBUR	Fiber Strand Buried	6 fiber 12 " 24 "	Fiber cable, including protective sheathing and waterproofing, plowed, bored, or laid directly in a trench in the earth and then covered. Includes associated engineering costs and material loading.	Sheath foot
FIUND	Fiber Strand Underground	6 fiber 12 " 24 "	Fiber cable that is placed in the underground conduit system. Includes associated engineering costs and material loading.	Sheath foot

MATERIAL TYPE CODE	MATERIAL TYPE	SIZES		DESCRIPTION	UNIT
LC4AR	Digital Loop Carrier 4 wire Basic Get Started Retail	24 48 96 192 224 448	672 896 1120 1344 1568 2016	The Retail 4-Wire Getting Started equipment investment for the Digital Loop Carrier. This equipment is a per DLC investment. The DLC size is based on the demand within the cluster being served. Includes material loading.	Per DLC
LC4AW	Digital Loop Carrier 4 wire Basic Get Started Wholesale	24 48 96 192 224 448	672 896 1120 1344 1568 2016	The Wholesale 4-Wire Getting Started equipment investment for the Digital Loop Carrier. This equipment is a per DLC investment. The DLC size is based on the demand within the cluster being served. Includes material loading.	Per DLC
LC4BR	Digital Loop Carrier 4 wire Line Card Retail	24 48 96 192 224 448	672 896 1120 1344 1568 2016	Plug-in equipment investment for the 4-wire Retail line. This investment will be applied to each loop being served from this DLC. Includes material loading and maintenance spares.	Per Line
LC4BW	Digital Loop Carrier 4 wire Line Card Wholesale	24 48 96 192 224 448	672 896 1120 1344 1568 2016	Plug-in equipment investment for the 4-wire Wholesale line. This investment will be applied to each loop being served from this DLC. Includes material loading and maintenance spares.	Per Line
LC4CR	Digital Loop Carrier 4 wire extended Retail	24 48 96 192 224 448	672 896 1120 1344 1568 2016	Plug-in investment for the 4-wire Extended Retail line. As there exists no 4-wire extended line card, this material type is a placeholder only.	Per Line Delta between Extended & Non-Extended Card

MATERIAL TYPE CODE	MATERIAL TYPE	SIZES		DESCRIPTION	UNIT
LC4CW	Digital Loop Carrier 4 wire extended Wholesale	24 48 96 192 224 448	672 896 1120 1344 1568 2016	Plug-in investment for the 4-wire Extended Wholesale line. As there exists no 4-wire extended line card, this material type is a placeholder only.	Per Line Delta between Extended & Non-Extended Card
LCBAR	Digital Loop Carrier Basic Getting Started – Retail	24 48 96 192 224 448	672 896 1120 1344 1568 2016	The Retail Basic Getting Started equipment investment for the Digital Loop Carrier. This equipment is a per DLC investment. The DLC size is based on the demand within the cluster being served. Includes material loading.	Per DLC
LCBAW	Digital Loop Carrier Basic Getting Started – Wholesale	24 48 96 192 224 448	672 896 1120 1344 1568 2016	The Wholesale Basic Getting Started equipment investment for the Digital Loop Carrier. This equipment is a per DLC investment. The DLC size is based on the demand within the cluster being served. Includes material loading.	Per DLC
LCBBR	Digital Loop Carrier Basic Line – Retail	24 48 96 192 224 448	672 896 1120 1344 1568 2016	Plug-in equipment investment for the Basic Retail line. This investment will be applied to each loop being served from this DLC. Includes material loading and maintenance spares.	Per Line
LCBBW	Digital Loop Carrier Basic Line – Wholesale	24 48 96 192 224 448	672 896 1120 1344 1568 2016	Plug-in equipment investment for the Basic Wholesale line. This investment will be applied to each loop being served from this DLC. Includes material loading and maintenance spares.	Per Line

MATERIAL TYPE CODE	MATERIAL TYPE	SIZES		DESCRIPTION	UNIT
LCBCR	Digital Loop Carrier Basic Extended Line - Retail	24 48 96 192 224 448	672 896 1120 1344 1568 2016	Plug-in investment for the Basic Extended Retail line. This DLC Line Card is utilized to serve Basic POTS customers beyond the 12 Kft serving area up to a maximum distance of 18 Kft. The DLC line card is used only when 18 Kft copper loop length is selected under Options/User.	Per Line Delta between Extended & Non-Extended Card
LCBCW	Digital Loop Carrier Basic Extended Line – Wholesale	24 48 96 192 224 448	672 896 1120 1344 1568 2016	Plug-in investment for the Basic Extended Wholesale line. This DLC Line Card is utilized to serve Basic POTS customers beyond the 12 Kft serving area up to a maximum distance of 18 Kft. The DLC line card is used only when 18 Kft copper loop length is selected under Options/User.	Per Line Delta between Extended & Non-Extended Card
LCCAR	Digital Loop Carrier Coin Get Started – Retail	24 48 96 192 224 448	672 896 1120 1344 1568 2016	The Retail Coin Getting Started equipment investment for the Digital Loop Carrier. This equipment is a per DLC investment. The DLC size is based on the demand within the cluster being served. Includes material loading.	Per DLC
LCCAW	Digital Loop Carrier Coin Get Started – Wholesale	24 48 96 192 224 448	672 896 1120 1344 1568 2016	The Wholesale Coin Getting Started equipment investment for the Digital Loop Carrier. This equipment is a per DLC investment. The DLC size is based on the demand within the cluster being served. Includes material loading.	Per DLC
LCCBR	Digital Loop Carrier Coin Line – Retail	24 48 96 192 224 448	672 896 1120 1344 1568 2016	Plug-in equipment investment for the Coin Retail line. This investment will be applied to each loop being served from this DLC. Includes material loading and maintenance spares.	Per Line

MATERIAL TYPE CODE	MATERIAL TYPE	SIZES		DESCRIPTION	UNIT
LCCBW	Digital Loop Carrier Coin Line – Wholesale	24 48 96 192 224 448	672 896 1120 1344 1568 2016	Plug-in equipment investment for the Coin Wholesale line. This investment will be applied to each loop being served from this DLC. Includes material loading and maintenance spares.	Per Line
LCCCR	Digital Loop Carrier Coin Extended Line – Retail	24 48 96 192 224 448	672 896 1120 1344 1568 2016	Plug-in investment for the Coin Extended Retail line. As there exists no Coin extended line card, this material type is a placeholder only.	Per Line Delta between Extended & Non-Extended Card
LCCCW	Digital Loop Carrier Coin Extended Line – Wholesale	24 48 96 192 224 448	672 896 1120 1344 1568 2016	Plug-in investment for the Coin Extended Wholesale line. As there exists no Coin extended line card, this material type is a placeholder only.	Per Line Delta between Extended & Non-Extended Card
LCDAR	Digital Loop Carrier Digital Data Services Get Started – Retail	24 48 96 192 224 448	672 896 1120 1344 1568 2016	The Retail DDS Getting Started equipment investment for the Digital Loop Carrier. This equipment is a per DLC investment. The DLC size is based on the demand within the cluster being served. Includes material loading.	Per DLC
LCDAW	Digital Loop Carrier Digital Data Services Get Started – Wholesale	24 48 96 192 224 448	672 896 1120 1344 1568 2016	The Wholesale DDS Getting Started equipment investment for the Digital Loop Carrier. This equipment is a per DLC investment. The DLC size is based on the demand within the cluster being served. Includes material loading.	Per DLC

MATERIAL TYPE CODE	MATERIAL TYPE	SIZES		DESCRIPTION	UNIT
LCDBR	Digital Loop Carrier Digital Data Services Line – Retail	24 48 96 192 224 448	672 896 1120 1344 1568 2016	Plug-in equipment investment for the DDS Retail line. This investment will be applied to each loop being served from this DLC. Includes material loading and maintenance spares.	Per Line
LCDBW	Digital Loop Carrier Digital Data Services Line – Wholesale	24 48 96 192 224 448	672 896 1120 1344 1568 2016	Plug-in equipment investment for the DDS Wholesale line. This investment will be applied to each loop being served from this DLC. Includes material loading and maintenance spares.	Per Line
LCDCR	Digital Loop Carrier Digital Data Services Extended Line – Retail	24 48 96 192 224 448	672 896 1120 1344 1568 2016	Plug-in investment for the DDS Extended Retail line. As there exists no DDS extended line card, this material type is a placeholder only.	Per Line Delta between Extended & Non-Extended Card
LCDCW	Digital Loop Carrier Digital Data Services Extended Line – Wholesale	24 48 96 192 224 448	672 896 1120 1344 1568 2016	Plug-in investment for the DDS Extended Wholesale line. As there exists no DDS extended line card, this material type is a placeholder only.	Per Line Delta between Extended & Non-Extended Card
LCIAR	Digital Loop Carrier Integrated Services Digital Network (ISDN) Get Started – Retail	24 48 96 192 224 448	672 896 1120 1344 1568 2016	The Retail ISDN Getting Started equipment investment for the Digital Loop Carrier. This equipment is a per DLC investment. The DLC size is based on the demand within the cluster being served. Includes material loading.	Per DLC

MATERIAL TYPE CODE	MATERIAL TYPE	SIZES		DESCRIPTION	UNIT
LCIAW	Digital Loop Carrier ISDN. Get Started – Wholesale	24 48 96 192 224 448	672 896 1120 1344 1568 2016	The Wholesale ISDN Getting Started equipment investment for the Digital Loop Carrier. This equipment is a per DLC investment. The DLC size is based on the demand within the cluster being served. Includes material loading.	Per DLC
LCIBR	Digital Loop Carrier (ISDN) Line – Retail	24 48 96 192 224 448	672 896 1120 1344 1568 2016	Plug-in equipment investment for the ISDN Retail line. This investment will be applied to each loop being served from this DLC. Includes material loading and maintenance spares.	Per Line
LCIBW	Digital Loop Carrier Integrated Service Network Line – Wholesale	24 48 96 192 224 448	672 896 1120 1344 1568 2016	Plug-in equipment investment for the ISDN Wholesale line. This investment will be applied to each loop being served from this DLC. Includes material loading and maintenance spares.	Per Line
LCICR	Digital Loop Carrier ISDN Extended Line – Retail	24 48 96 192 224 448	672 896 1120 1344 1568 2016	Plug-in investment for the ISDN Extended Retail line. As there exists no ISDN extended line card, this material type is a placeholder only.	Per Line Delta between Extended & Non-Extended Card
LCICW	Digital Loop Carrier ISDN Extended Line – Wholesale	24 48 96 192 224 448	672 896 1120 1344 1568 2016	Plug-in investment for the ISDN Extended Wholesale line. As there exists no ISDN extended line card, this material type is a placeholder only.	Per Line Delta between Extended & Non-Extended Card
MDF	Main Distribution Frame	1		A metal framework used for mounting the protectors (termination of the outside cable) on one side and the termination blocks for jumper cables from the switch and other central office components on the other side. Jumper wires are placed from one side to the other to complete connections.	Per Pair

MATERIAL TYPE CODE	MATERIAL TYPE	SIZES	DESCRIPTION	UNIT
MDFPROT	Main Distribution Frame Protector	1	The protection device located in the wire center, on the Main Distribution Frame, that serves as a lightning suppresser to protect the central office equipment.	Per Line
MMPOLE	Material loading for Poles	1	N/A Input code no longer applicable.	
MNHOLE	Manhole	1	A prefabricated 5'10"x10'6"x6'6" concrete manhole. Includes associated engineering costs and material loading.	Per manhole
MUX	Multiplexer	31	The electronic equipment used in smaller switch nodes as an economical means to multiplex and demultiplex between DS-3 and DS-1. Includes material loading.	Per multiplexer
NACDSXJ		0 1 3	N/A Input code no longer applicable.	
NID	Network Interface Device Housing	1	6 pair NID housing: A network interface device housing with 6 pair capacity used to terminate the telephone company's facilities (drop wire) at the customer's location. This is also the interface device between the customer's inside wiring and the telephone network.	Per NID
NIDPROT	Network Interface Device Protector Module	1	NID protector modules for use in 6 pair NID housings. Each protector unit provides protection for one pair from hazardous voltages and currents and separates the customer's equipment from the rest of the local network. Includes material loading with the exception of costs associated with minor materials.	Per protector module
NID	Network Interface Device	12 25 50	A network interface device housing with 12, 25 or 50 pair capacity wired between a local carrier's network, where the drop wire terminates, and the inside wiring of a customer location. The 12 and 25 pair, and 50 pair module, housing, cover and clip include the material loading with the exception of costs associated with minor materials. 50 pair panel costs include the material loading.	Per NID
NIDPROT	Network Interface Device Protector Module	12 25 50	NID protector modules for use in 12, 25, and 50 pair NID housings. Each protector unit provides protection for one pair from hazardous voltages and currents and separates the customer's equipment from the rest of the local network. Includes material loading with the exception of costs associated with minor materials.	Per protector module
OC		3 12	The plug-in cards associated with ADMs used to inter-network two ADM optically without having to convert the optical signal to its electrical equivalent. The OC3 card inter-networks the terminals at an OC-3 rate, and the OC12 card inter-networks the terminals at an OC-12 rate. Includes material loading	Per Card
POLE	Telephone Pole	1	The material cost (not including labor) of a 30' Class 5 treated wood utility pole. Includes associated engineering costs and material loading.	Per pole
POLESH	Telephone Pole	1	The material cost (not including labor) of a 40' Class 4 treated wood utility pole. Includes associated engineering costs and material loading.	Per pole

MATERIAL TYPE CODE	MATERIAL TYPE	SIZES		DESCRIPTION	UNIT
PULLBOX	Pull box	1		4'x4'x4' Handhole: A pre-fabricated 4'x4'x4' small concrete handhole. Includes associated engineering costs and material loading.	Per pull box
STRAND	10m Strand	1		The steel suspension strand, also referred to as a "messenger", that provides structural support to aerial cable between the poles. The aerial cable is usually attached to the strand by wire wrapped around the cable and strand. The strand also provides electrical continuity throughout the aerial exchange cable network to protect against direct contacts and lightning hazards when properly attached to the multi-ground neutral. Includes associated engineering costs and material loading with the exception of costs associated with minor materials.	Linear foot
SUBDUC	Subduct	1		A 1" or 1 ¼ " PVC pipe through which fiber cables can be passed. Two or three subducts are usually placed in a 4" conduit to allow 2 or 3 fiber cables to be placed in one 4" duct. The subduct can also be buried directly in a trench to facilitate fiber cable placement. Includes associated engineering costs and material loading with the exception of costs associated with minor materials.	Linear foot
TERM	Pedestal Terminal	1		The loop component that serves as the access point between the distribution cable and the drop wire. The distribution terminal, in this case, is located in a pedestal and is spliced to pairs in the distribution cable to make them available for use to serve the customer. Includes associated engineering costs and material loading with the exception of costs associated with minor materials.	Per terminal
XCONN	Cross-Connect Cabinet	400 600 900 1800	2700 3600 5400	A cabinet containing termination blocks on which the feeder and distribution cable pairs are both terminated. Each type of cable is usually terminated in a designated area. Jumper wires are used to connect the assigned feeder pair to the appropriate distribution pair. The cross connect box provides flexibility (any feeder pair can be connected to any distribution pair) and allows the feeder and distribution cables to be sized differently. Distribution cables are sized for the ultimate demand while feeder cables are usually sized for 3 to 5 years growth. Includes associated engineering costs and material loading.	Unit

Outside Plant Placement Costs [XXLABR.DB]

Purpose: Provides labor costs for installing specific types of equipment and facilities. The labor requirements and tasks included in each cost are defined in the OSP Placement table below. In states where there are significant cost differences between areas, two sets of weighted average rates are developed (Tier A and Tier B).

COLUMN HEADINGS	DESCRIPTION
LABOR TYPE	Type of labor required, as identified by an alphanumeric code (for example, LP01A for placing a pole). The tasks involved in each type of labor are defined in the OSP Placement table below.
DESCRIPTION	A description of the work done (for example, "place anchor & guy"). These descriptions are extracted from the contracts used by GTE and the contractor. They specifically identify the tasks that are included in any labor requirement.
TIER A RATE	Labor activity rate for Tier A areas. Used by ICM when FALSE appears in the Tier B column in the XXNODES.DB table.
TIER B RATE	Labor activity rate for Tier B areas. Used by ICM when TRUE appears in the Tier B column of the XXNODES.DB table.

Outside Plant Placement Input Descriptions

Below is a table describing the outside plant placement inputs used in developing the investments for loop and transport components. The user can modify the prices associated with these activities and/or insert or delete inputs.

* Labor code conventions for Digital Loop Carrier (DLC) are as follows:

1st & 2nd Characters		3rd Character		4th Character		5th Character	
LC	Loop Carrier	B	Basic	A	Getting Started	R	Retail Price
		C	Coin	B	Per Line	W	Wholesale Price
		D	Digital Data Service	C	Loop Extender		
		I	Integrated Services Digital Network (ISDN)				
		4	4 Wire				

LABOR TYPE CODE	LABOR TYPE	SIZES		DESCRIPTION	UNIT
LADM	Install ADM equipment	3 12 48		Includes Engineering and Installation Labor for the placement of the OC-3, OC-12, and OC-48 ADM Common Equipment.	Per ADM
LC4AR	Install 4 wire DLC Get Started Retail	24 48 96 192 224 448	672 896 1120 1344 1568 2016	The Retail 4-Wire Getting Started labor investment for the Digital Loop Carrier. This labor is a per DLC investment. Includes the Engineering and Installation labor for the placement of the DLC Getting Started equipment.	Per DLC
LC4AW	Install 4 wire DLC Get Started Wholesale	24 48 96 192 224 448	672 896 1120 1344 1568 2016	The Wholesale 4-Wire Getting Started labor investment for the Digital Loop Carrier. This labor is a per DLC investment. Includes the Engineering and Installation labor for the placement of the DLC Getting Started equipment.	Per DLC

LABOR TYPE CODE	LABOR TYPE	SIZES		DESCRIPTION	UNIT
LC4BR	Install 4 Wire DLC Retail	24 48 96 192 224 448	672 896 1120 1344 1568 2016	Plug-in labor investment for the 4-wire Retail line. This investment will be applied to each loop being served from this DLC. Includes the Engineering and Installation labor for the placement of the DLC Plug-In card.	Per Line
LC4BW	Install 4 Wire DLC Wholesale	24 48 96 192 224 448	672 896 1120 1344 1568 2016	Plug-in labor investment for the 4-wire Wholesale line. This investment will be applied to each loop being served from this DLC. Includes the Engineering and Installation labor for the placement of the DLC Plug-In card.	Per Line
LCB	Install Channel Bank			Includes Engineering and Installation Labor for the placement of the Channel Bank Common Equipment.	Per CB
LCBAR*	Install DLC Get Started-Retail	24 48 96 192 224 448	672 896 1120 1344 1568 2016	The Retail Basic Getting Started labor investment for the Digital Loop Carrier. This labor is a per DLC investment. Includes the Engineering and Installation labor for the placement of the DLC Getting Started equipment.	Per DLC
LCBAW	Install DLC Get Started-wholesale	24 48 96 192 224 448	672 896 1120 1344 1568 2016	The Wholesale Basic Getting Started labor investment for the Digital Loop Carrier. This labor is a per DLC investment. Includes the Engineering and Installation labor for the placement of the DLC Getting Started equipment.	Per DLC
LCBBR	Install DLC Line-retail	24 48 96 192 224 448	672 896 1120 1344 1568 2016	Plug-in labor investment for the Basic Retail line. This investment will be applied to each loop being served from this DLC. Includes the Engineering and Installation labor for the placement of the DLC Plug-In card.	Per Line

LABOR TYPE CODE	LABOR TYPE	SIZES		DESCRIPTION	UNIT
LCBBW	Install DLC Line-wholesale	24 48 96 192 224 448	672 896 1120 1344 1568 2016	Plug-in labor investment for the Basic Wholesale line. This investment will be applied to each loop being served from this DLC. Includes the Engineering and Installation labor for the placement of the DLC Plug-In card.	Per Line
LCBCARD	Install Channel Bank Card			Includes Engineering and Installation Labor for the placement of the Channel Unit	Per unit
LCCAR	Install DLC Get Started-retail	24 48 96 192 224 448	672 896 1120 1344 1568 2016	The Retail Coin Getting Started labor investment for the Digital Loop Carrier. This labor is a per DLC investment. Includes the Engineering and Installation labor for the placement of the DLC Getting Started equipment.	Per DLC
LCCAW	Install DLC Get Started-wholesale	24 48 96 192 224 448	672 896 1120 1344 1568 2016	The Wholesale Coin Getting Started labor investment for the Digital Loop Carrier. This labor is a per DLC investment. Includes the Engineering and Installation labor for the placement of the DLC Getting Started equipment.	Per DLC
LCCBR	Install DLC Line-retail	24 48 96 192 224 448	672 896 1120 1344 1568 2016	Plug-in labor investment for the Coin Retail line. This investment will be applied to each loop being served from this DLC. Includes the Engineering and Installation labor for the placement of the DLC Plug-In card.	Per Line
LCCBW	Install DLC Line-wholesale	24 48 96 192 224 448	672 896 1120 1344 1568 2016	Plug-in labor investment for the Coin Wholesale line. This investment will be applied to each loop being served from this DLC. Includes the Engineering and Installation labor for the placement of the DLC Plug-In card.	Per Line

LABOR TYPE CODE	LABOR TYPE	SIZES		DESCRIPTION	UNIT
LCDAR	Install DLC Get Started-retail	24 48 96 192 224 448	672 896 1120 1344 1568 2016	The Retail DDS Getting Started labor investment for the Digital Loop Carrier. This labor is a per DLC investment. Includes the Engineering and Installation labor for the placement of the DLC Getting Started equipment.	Per DLC
LCDAW	Install DLC Get Started-wholesale	24 48 96 192 224 448	672 896 1120 1344 1568 2016	The Wholesale DDS Getting Started labor investment for the Digital Loop Carrier. This labor is a per DLC investment. Includes the Engineering and Installation labor for the placement of the DLC Getting Started equipment.	Per DLC
LCDBR	Install DLC Line-retail	24 48 96 192 224 448	672 896 1120 1344 1568 2016	Plug-in labor investment for the DDS Retail line. This investment will be applied to each loop being served from this DLC. Includes the Engineering and Installation labor for the placement of the DLC Plug-In card.	Per Line
LCDBW	Install DLC Line-wholesale	24 48 96 192 224 448	672 896 1120 1344 1568 2016	Plug-in labor investment for the DDS Wholesale line. This investment will be applied to each loop being served from this DLC. Includes the Engineering and Installation labor for the placement of the DLC Plug-In card.	Per Line
LCIAR	Install DLC Get Started-retail	24 48 96 192 224 448	672 896 1120 1344 1568 2016	The Retail ISDN Getting Started labor investment for the Digital Loop Carrier. This labor is a per DLC investment. Includes the Engineering and Installation labor for the placement of the DLC Getting Started equipment.	Per DLC

LABOR TYPE CODE	LABOR TYPE	SIZES		DESCRIPTION	UNIT
LCIAW	Install DLC Get Started-wholesale	24 48 96 192 224 448	672 896 1120 1344 1568 2016	The Wholesale ISDN Getting Started labor investment for the Digital Loop Carrier. This labor is a per DLC investment. Includes the Engineering and Installation labor for the placement of the DLC Getting Started equipment.	Per DLC
LCIBR	Install DLC Line-retail	24 48 96 192 224 448	672 896 1120 1344 1568 2016	Plug-in labor investment for the ISDN Retail line. This investment will be applied to each loop being served from this DLC. Includes the Engineering and Installation labor for the placement of the DLC Plug-In card.	Per Line
LCIBW	Install DLC Line-wholesale	24 48 96 192 224 448	672 896 1120 1344 1568 2016	Plug-in labor investment for the ISDN Wholesale line. This investment will be applied to each loop being served from this DLC. Includes the Engineering and Installation labor for the placement of the DLC Plug-In card.	Per Line
LDCS1	Install DCS1 equipment			Includes Engineering and Installation Labor for the placement of the DCS1/0 Common Equipment.	Per DACS
LDCS11P	DCS1/1 Port Eng/Install			Includes Engineering and Installation Labor for the placement of the DCS1/0 DS1 ports.	Per port
LDCS3	Install DCS3 equipment			Includes Engineering and Installation Labor for the placement of the DCS3/1 Common Equipment.	Per DACS
LDCS31P	DCS3/1 Port Eng/Install			Includes Engineering and Installation Labor for the placement of the DCS3/1 DS1 ports.	Per port
LDCS33P	DCS3/3 Port Eng/Install			Includes Engineering and Installation Labor for the placement of the DCS3/1 DS3 ports.	Per port
LDS1	Install DS1 card			The ADM plug-in card Engineering and Installation labor investment required for the placement of the DS-1 Port Card.	Per Card
LDS3	Install DS3 card			The ADM plug-in card Engineering and Installation labor investment required for the placement of the DS-3 Port Card.	Per Card
LDSX1	Install DSX1 equipment			Includes Engineering and Installation Labor for the placement of DSX-1 Cross Connect panel.	Per panel
LDSX1J	DSX-1 Jumper Eng/Install			Includes Installation Labor for the placement of DSX-1 Cross Connect jumper.	Per jumper

LABOR TYPE CODE	LABOR TYPE	SIZES	DESCRIPTION	UNIT
LDSX3	Install DSX3 equipment		Includes Installation Labor for the placement of DSX-3 Cross Connect panel.	Per panel
LDSX3J	DSX-3 Jumper Eng/Install		Includes Installation Labor for the placement of DSX-3 Cross Connect module and jumper.	Per Jumper
LFDM	Install fiber distribution module		Includes Engineering and Installation Labor for the placement of a Fiber Distribution Panel Module.	Per Module
LFDP	Install Fiber Distribution Panel		Includes Engineering and Installation Labor for the placement of a Fiber Distribution Panel common equipment.	Per panel
LMDFPROT	Install MDF Protector		The Installation labor investment required for the placement of Protection on the Main Distribution Frame in the central office.	Per Line
LMUX3	Install multiplexer		Includes Engineering and Installation Labor for the placement of DS3/DS1 Multiplexer.	Per Mux
LOC3CARD	Install OC3 card		The ADM plug-in card Engineering and Installation labor investment required for the placement of the OC-3 inter-networking card.	Per Card
LOC12CARD	Install OC12 card		The ADM plug-in card Engineering and Installation labor investment required for the placement of the OC-12 inter-networking card.	Per Card
LP01A	Placing Pole		Includes transporting and setting pole at the proper location, depth, and alignment, compacting distributed soil (tamping), and tagging (pole number). Also includes stepping pole and disposing of surplus dirt or rock where required, and placing a butt ground when required.	Per pole
LP03A	Replaces LP01A for shared pole		This unit applies if a pole is placed, removed, straightened, or reset in a power line when either a safety blanket or the presence of the power company is required. This unit includes all of the operations described in placing, removing, and straightening poles.	Per pole
LP07A	Place anchor & guy		Includes installing the anchor and rod(s) to the proper depth, placing and tensioning down guy and sidewalk guy, and installing guy guards. Removal of rock is included where necessary.	Per anchor
LP18A	Place strand		This covers the placement of strand, hardware, down guys, tensioning of strand, and placing ground rods/ wires as required.	Linear Foot
LP22A	Place aerial cable		This covers all handling associated with placing aerial cable/sub-duct/cable in sub-duct on existing strand or overlashing with existing cable. Includes double lashing or delash/relash (where required), placing wire clamps, spacers, straps, cable dampers, tree/squirrel guards, riser/U-guards as required, placement of fiber tags, and any incidental tree trimming. Also includes retensioning of the existing strand and placing additional down guys, if required, to meet specifications.	Per foot

LABOR TYPE CODE	LABOR TYPE	SIZES	DESCRIPTION	UNIT
LP28A	Place cross-connect		This covers placement of all cross-connects, either pad or pole mounted. It includes forming the stub of a pre-stubbed cabinet, placing U-guard, supports, and spacers as required, up the pole or through conduit to a remote switch unit in close proximity. Also includes proper bonding and grounding, and tagging (numbering) the cabinet.	Per cross-connect
LP28D	Place xconnect pad		This covers the installation of a preformed pad (furnished by GTE). Includes preparing the base, placing grounding materials and may include placement of up to 20 feet of conduit with up to 6 bends as required.	Per pad
LP43A	Place copper cable in conduit		This includes transportation and placement of metallic cable in conduit (including direct-buried conduit without both ends exposed) and into or through manholes, vaults, or riser locations when pull line is in place. It includes manhole set-up, placing the pull line if required, proper racking of the cable in manhole or vault, and placing cable up risers with clamps/guards, etc. Includes proofing the duct and the mandatory pressure checks to ensure that the cable has not been damaged during placing.	Per cable foot
LP43C	Place fiber cable in conduit		This covers placement of fiber optic cable in duct, and includes all handling of the cable to avoid cutting of the cable except as specified by GTE engineering, proper racking, and placing split sub-duct over the cable in manholes or vaults. Manhole set-up, placing pull line, and proofing the duct are also included.	Per cable foot
LP49A	Place cable/duct direct		Includes all transportation, handling, and installation of one or more cables and/or a single duct in an open trench provided by GTE. Placing riser, U-guards, cable marker stakes/posts, and ID tape/locating conductors.	Per trench foot
LP49B	Place add cable/duct		Includes transportation, handling, and installation of an additional cable/duct or sub-duct in an open trench provided by the Contractor, including placement of risers and/or U-guards.	Per trench foot
LP50A	Place buried drop		Covers all work, including trenching, required to place one or more direct buried service wires, or service wires in duct, or the duct itself. The unit includes placement of the service wire or duct at the appropriate depth (a minimum of 12 inches), proper routing of the service wire to the serving terminal, leaving adequate slack on a pole, stake or cane as required for termination, backfill of all excavations. It does NOT include placing a protector/network interface device (NID/SNID, etc.). Manual tunneling is considered incidental and will not be reimbursed by GTE.	Per trench foot
LP51A	Plow cable-jobs < 1000 units		This applies to plowing all types and size of buried cable and/or sub-duct at a depth of 30". Also includes incidental hand digging/backhoeing to expose existing substructures, or to extend trench to a pedestal or pole or any other necessary digging required for placement of cable, backfilling, compaction and restoring the property to it's original condition, placing all risers/U-guards, cable marker stakes/signs, and ID tape/locating conductors.	Per linear foot

LABOR TYPE CODE	LABOR TYPE	SIZES	DESCRIPTION	UNIT
LP51B	Plow cable-jobs > 1000 units		See "Plow cable-jobs" < 1000 (LP51A) above. Charged per linear foot.	Per linear foot
LP51C	Plow Additional 6"		This should be used in conjunction with Plow cable-jobs when additional depth is required.	Per plowed foot
LP52A	Pre-ripping		This includes preripping with a cable plow when required by adverse soil conditions to allow plowing in of buried cable and/or sub-duct.	Per linear foot
LP54A	Trench @ 30"- JOB <1000		This includes all labor and equipment required to open a trench with a trenching machine and placement of a buried cable or single conduit at a depth of 30". Also included is incidental hand digging/ backhoeing to expose existing substructures, or to extend trench to a pedestal or pole (or any other necessary digging required for placement of cable), backfilling, compaction and restoring the property to its original condition, placing all riser/U-guards, cable marker stakes/signs, and ID tape/locating conductors.	Per linear foot
LP54B	Trench @ 30"- Job >1000		See Trench @ 30"- JOB <1000 (LP54A) above. Charged per linear foot.	Per linear foot
LP54C	Increase trench depth by 6"		This is to be used in conjunction with Trench @ 30"- Job when additional depth is required.	Per trench foot
LP55A	Backhoe @36" & place cable/con		Includes all transportation, handling, and installation of one or more cables and/or a single conduit in an open trench, 36" in depth, with the use of a backhoe. Placing riser, U-guards, cable marker stakes/posts, and ID tape/locating conductors.	Per trench foot
LP55B	Increase backhoe depth by 12""		Provides for an increased depth of 12", to be used in conjunction with Backhoe @36".	Per trench foot
LP57A	Hand dig trench		Includes manual removal of soil when common machinery is not the most efficient or effective method of digging a trench.	Per trench foot
LP59A	Initial bore		This applies to a successful bore or pipe push. Incidental is any associated digging, pulling through cable(s)/sub-duct, locating the end of the pipe and marking the ends (if required), backfilling, compacting, and restoring the property to its original condition. It does NOT include the cost of the pipe placed that will be provided by GTE or billed separately.	Per linear foot
LP61A	Place pedestal		This includes the placement of an above ground pedestal. It includes placing a stake/pole mounted CAD pedestal, bringing new cable(s) into the unit, placing gravel, ground wire, attaching the ground wire to an existing MGN or placing ground rod as required, bonding and grounding, labeling and terminal decals.	Per pedestal
LP70A	Cut solid rock - per hole		This covers the cutting of solid rock during a placing operation. It includes backfilling the trench with packed soil and disposing of excess rock.	Per hole

LABOR TYPE CODE	LABOR TYPE	SIZES	DESCRIPTION	UNIT
LP70B	Rock saw		This covers the cutting of solid rock during a placing operation. It includes backfilling the trench with packed soil and disposing of excess rock.	Per linear foot
LP70C	Plow additional cable		This applies during the plowing of two cables or if a split duct is to provide additional protection to the cable (e.g., fiber optic cable placed in rocky soil).	Per linear foot
LP73A	Splice Pit		This covers digging of a splice pit, by hand or machine. A standard splice pit is approximately 4'x5'x1' below the cable. This includes getting all utility locates, digging the pit, providing a safety perimeter around the pit if required, backfilling, compacting for proper support under the splice closure, and restoral of the surface.	Per pit
LP73B	Well Point		This will apply when well pointing is authorized due to high water table conditions. This will apply when digging a splice pit, or when providing well point(s) at 50 foot intervals for trench line. It includes any additional necessary digging as well as the installation and monitoring of the de-watering system.	Per location per day
LP87A	Rod and mandrel duct		This is used for the combination activity of rodding, mandrelling, placement of pull line and verifying end to end measurements in preparation for placement of underground cable in an existing conduit. Includes manhole set-up.	Per duct/foot
LP88A	Place sub-duct or air tube in conduit		This covers the hauling of sub-duct/air tube and associated material necessary to install sub-duct or air tube in a conduit, placing the sections together, and properly forming and securing the sub-duct/air tube to manhole walls and/or racks in manhole or cable vaults. Includes manhole set-up, and placing pull line if required.	Per duct foot
LP93C	Cut and remove concrete		This includes all labor and equipment required to cut and remove concrete by any method and haul it from the job site. It also includes backfilling the excavation with dirt, compaction of the dirt and smoothing of the surface.	Per square foot of surface
LP93D	Place concrete		This includes all labor, equipment and delivery of materials required to place concrete to required thickness. Also included is preparation and compaction of the appropriate base material (sand, gravel, limestone rock, etc) prior to placing concrete as required, according to local city, county, or state laws. This doesn't include the cost of material.	Per square foot of surface
LS02A	Straight splice 1-50 pairs		Includes the permanent connecting of individual wires of a pair to those of another pair at the junction of two or more cables. Includes set-up and closure of cable. Also includes forming/racking of cables, manhole setup, tagging and pair identification.	Per pair
LS02B	Straight splice 51-300 pairs		See Straight splice 1-50 pairs (LS02A) above. Charge is per pair.	Per pair
LS02C	Straight splice +300 pairs		See Straight splice 1-50 pairs (LS02A) above. Charge is per pair.	Per pair

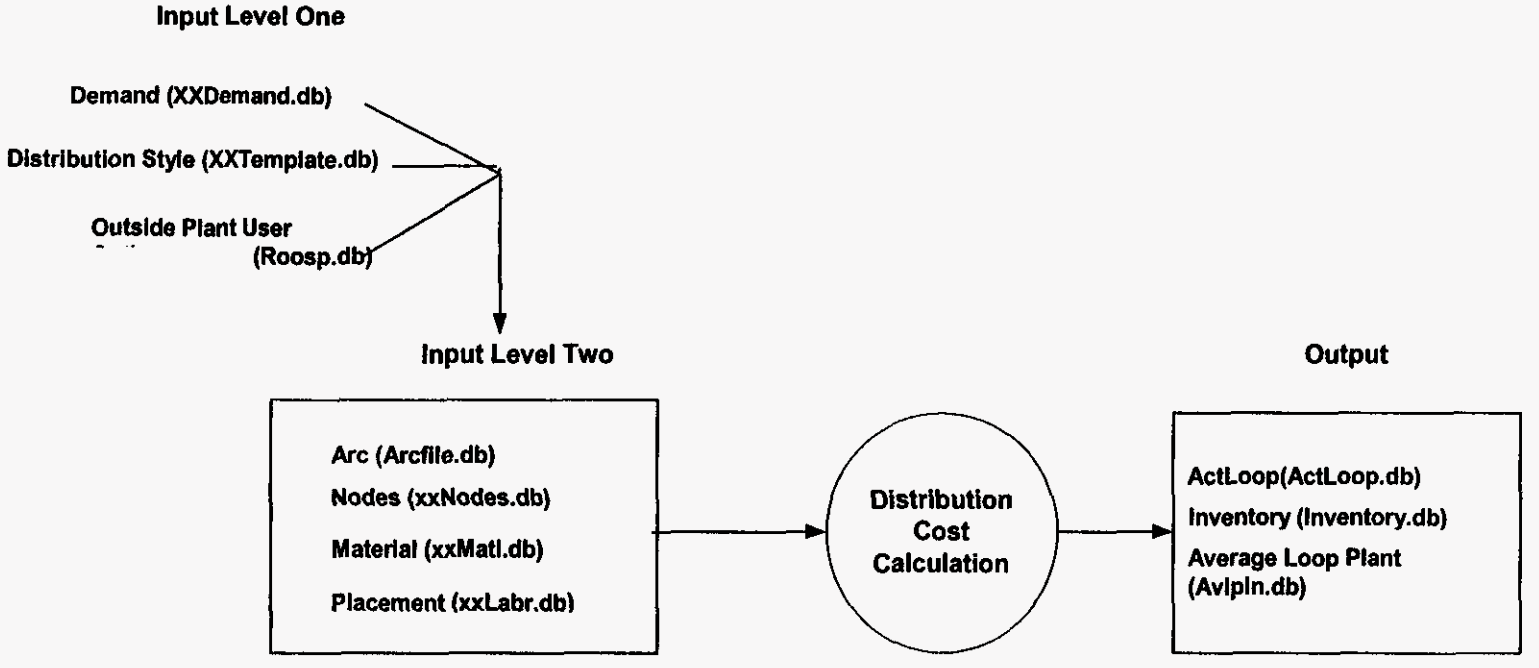
LABOR TYPE CODE	LABOR TYPE	SIZES	DESCRIPTION	UNIT
LS13A	Place fixed counted terminal		This covers the placement of any fixed count terminal (pedestal mounted, aerial, building, NID, etc.). Included is all set-up, associated splicing (straight/branch, etc.), placing seals and static stoppers, bonding and grounding as required, placing closure/ ped-caps, pea gravel/pest control (if not already completed), and labeling as required.	Per closure/ housing
LS14B	Place NID		This covers the placement of demarcation hardware (up to and including 6 pair) at a building or junction of aerial and buried plant, placement of static stoppers, and proper bonding and grounding according to GTE practices. If placing the interface device requires a trailer/mobile home stake, placement of the stake will be included in this unit (except in areas where stakes are provided by the customer.). This unit also includes placing, moving, or removing analog or digital station channel units or repeaters.	Per NID
LS19A	Setup pedestal		This includes opening/closing the sheath in a cable which is looped through a pedestal. It includes sheath removal, bonding and grounding of cable to the pedestal, placing binder group identification markers, ped caps, pea gravel/pest control (if not already completed), and numbering the pedestal as required.	Per pedestal
LS20A	Run jumpers at any cross connect		This covers placing, removing, or rearranging jumpers at any cross-connect location, including the central office.	Per jumper
LS26A	Conduit acceptance testing		This includes acceptance testing per GTE Practice and providing the necessary documentation of the results. It also includes Contractor provided equipment, set-up and testing.	Per pair
LS50A	Splice Fiber (48 fibers or less)		This covers the permanent connection of pigtails to a fiber of a cable or splicing a fiber of one cable to a fiber of another cable (mechanical or fusion) and associated testing. It includes sheath preparation and labeling, as required. Testing includes testing for the loss of individual splices to ensure they meet minimum requirements. It also involves end-to-end testing in both directions and documentation of the results. Includes set-up and closure.	Per fiber
LS50B	Splice Fiber (more than 48)		This covers the permanent connection of pigtails to a fiber of a cable or splicing a fiber of one cable to a fiber of another cable (mechanical or fusion) and associated testing. It includes sheath preparation and labeling, as required. Testing includes testing the loss of individual splices to ensure they meet minimum requirements. It also involves end-to-end testing in both directions and documentation of the results. Includes set-up and closure.	Per fiber
LS72A	Manhole setup for use		Included is all associated labor, material, and equipment required for preparing a manhole for entry, including but not limited to generator, water pump, blower, gas detector, manhole guard, truck and tools, and standard traffic control devices (signs and cones).	Per manhole

LABOR TYPE CODE	LABOR TYPE	SIZES	DESCRIPTION	UNIT
MNHOLE	Manhole install		This unit involves all work necessary for the installation and placement of a manhole. This unit includes the restoration of the surface in the immediate vicinity of the manhole with the appropriate material. If major restoration is required by local governing bodies, such as the entire side of a paved road, then this would be paid under other units. Manhole setup is included if required by GTE.	Per manhole
PT001	Install Cable to Interconnect Cage		This unit covers installation or removal of new or existing cable to any interconnect cage.	Per foot
PT002	Install DS0 Jumper @ Frame		This unit covers the installation or removal of a DS0 jumper to any designated frame	Per jumper
PT003	Install DS1 Jumper @ DSX		This unit covers the installation or removal of a DS1 jumper to any Digital Signal Cross Connect Box.	Per jumper
PT004	Install DS3 @ DSX		This unit covers the installation or removal of a DS3 jumper to any Digital Signal Cross Connect Box.	Per jumper
PT005	Install Conn Block @ Frame		This unit covers the installation of the connector block at the main distribution frame.	Per block
PULLBOX	Pull box installation		This unit involves all work necessary for the installation and placement of a pull box. This unit includes the restoration of the surface in the immediate vicinity of the manhole with the appropriate material. If major restoration is required by local governing bodies, such as the entire side of a paved road, then this would be paid under other units. Pull box setup is included if required by GTE. Note: Restoration of the surface will be limited to an area within five feet of the perimeter of the pull box lid.	Per box

Processing Flow

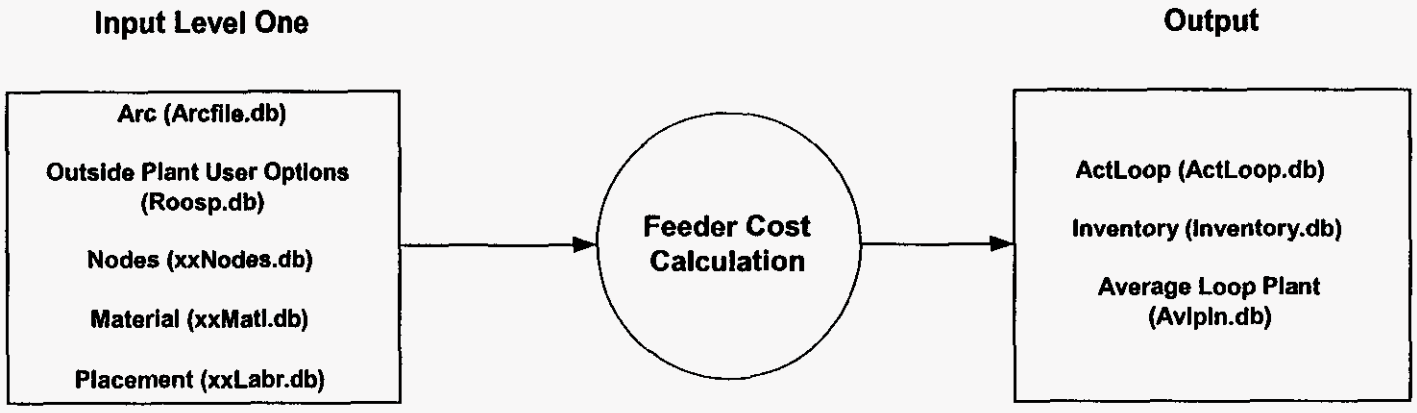
The Distribution bubble chart provides a graphic view of the database files that are input to and output of the Distribution portion of the Loop Module.

Distribution Bubble Chart



The Feeder bubble chart provides a graphic view of the database files that are input to and output of the Feeder portion of the Loop Module.

Feeder Bubble Chart

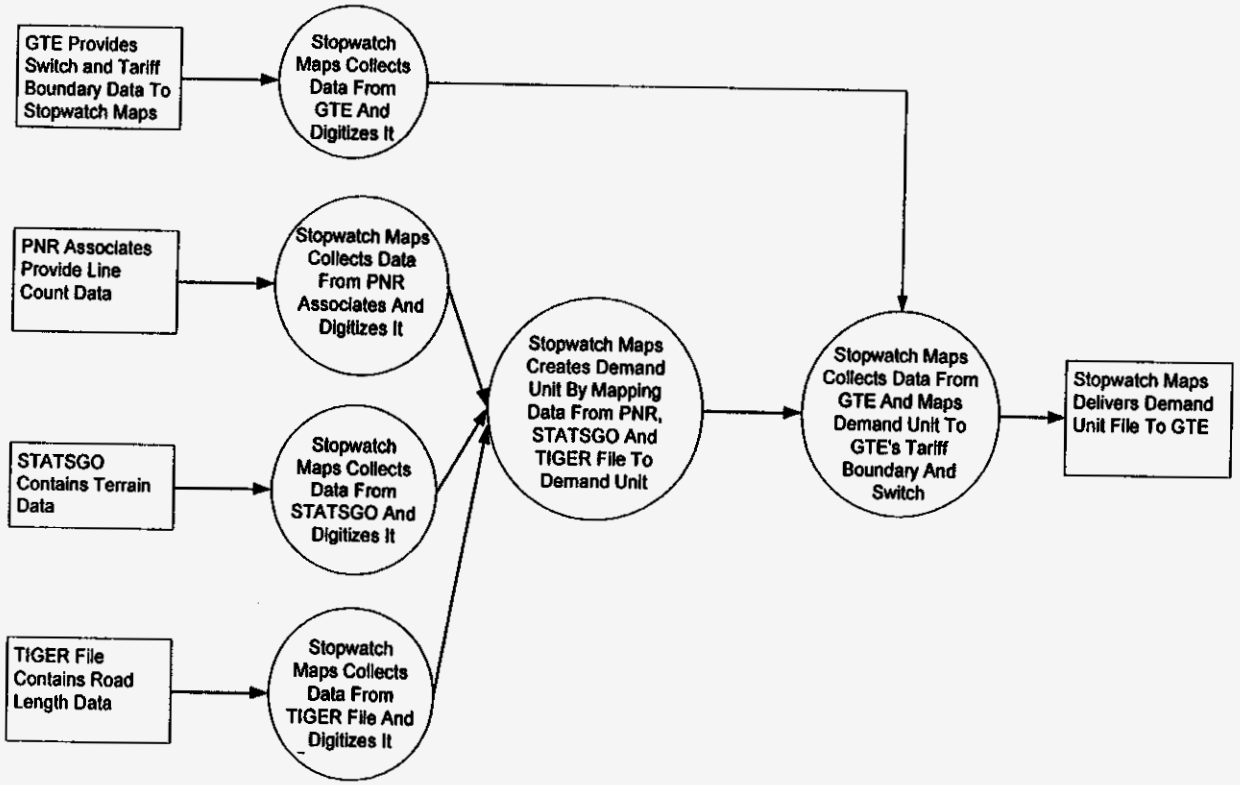


APPENDIX B: Stopwatch Maps' Role In Generating Demand Unit File

The basic building block of the ICM loop module is the 1/200th by 1/200th degree *demand unit*, each of which contains line counts (residential and business), road feet, and terrain conditions such as bedrock depth, and water table depth. All of the data mapped to the demand units is based on publicly available information. The lines counts, available at the census block level, are purchased from PNR Associates and are based on US Census data. The road information comes from the TIGER Files, and the terrain data comes from STATSGO. Roads and terrain data are purchased from Stopwatch Maps.

In addition to providing terrain and road data, Stopwatch Maps also distributes the various categories of information to demand units and assigns demand units to wire centers using a sophisticated Geographic Information System (GIS) software package. The software used by Stopwatch Maps allows the data, that has been collected by census block and is highly irregular in size and shape, to be accurately mapped to the standard 1/200th degree demand units. The flow charts below illustrate the flow from the raw inputs, through pre-processing, to standardized demand unit data outputs.

Demand Unit File Process Used By Stopwatch Maps

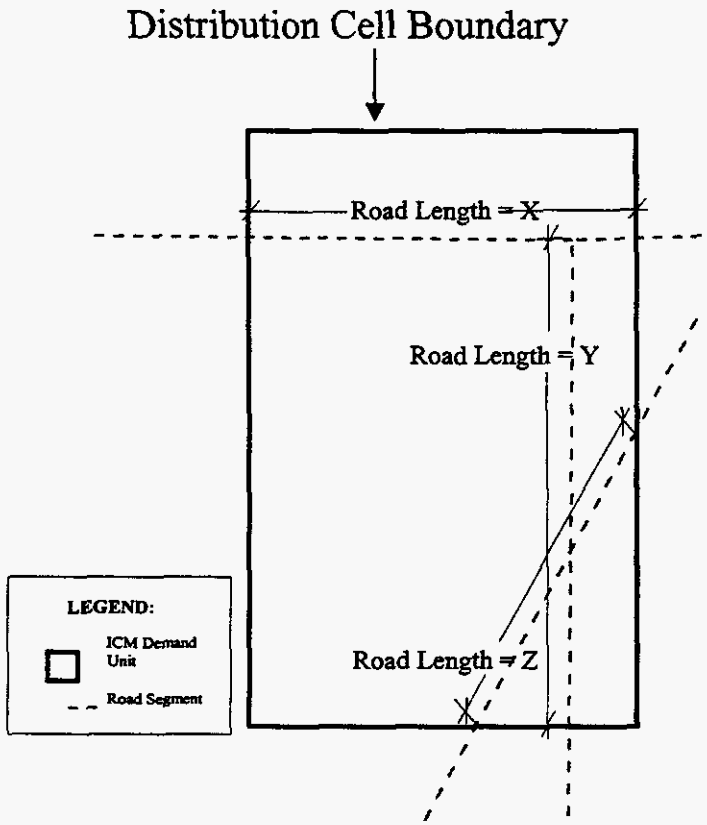


Stopwatch Maps extracts all road/street sections except limited access highways and their ramps and other minor categories of roads from the Tiger files into a MapInfo table for counties served by GTE. Limited access highways and their ramps are excluded because it is extremely unlikely that a house would be located along that type of road. Stopwatch Maps then overlays, through MapInfo, the relevant demand unit network upon this table, calculating the total length of road sections in demand unit.

The diagram below depicts three road segments within one demand unit. The sum of each identified road segment is assigned to the demand unit. This sample demand unit would result in a road length assignment of:

$$x + y + z = \textit{total demand unit road length in feet}$$

This calculation takes into account the length of curved and straight sections of road, but does not account for the distance required to go up and down hills. The resulting sum is rounded to a whole number.



The intention of the terrain development data process is to determine the overall surface character of any given demand unit, in order to better evaluate the potential costs involved in providing telephone service. The data utilized to support this process originates from the US Department of Agriculture, the State Soil Geographic Data Base (STATSGO), previously processed to produce the Stopwatch Maps product *State Terrain by Demand unit*, which was directly used in this effort.

Stopwatch Maps programmatically overlays the previously generated demand unit network and the terrain table. Bedrock Depth and Water Table Depth that vary within the area of a demand unit are averaged and assigned appropriately.

APPENDIX C: Distribution Styles & Distribution Diagrams

The following pages contain the nine distribution diagrams (or stick maps) that are currently used in ICM in the development of distribution investments. These diagrams/templates cover varied levels of demand and show how one can generate distribution diagrams of various shapes and sizes. These diagrams, however, are just pointers. A user may devise his/her own diagrams depending on his/her demand database and specific requirements.

Stick Map for Distribution Style One



In Distribution Style One, the backbone cables have two spans. Both the spans support only itself. Hence, in this case, both the spans are labeled as "A".

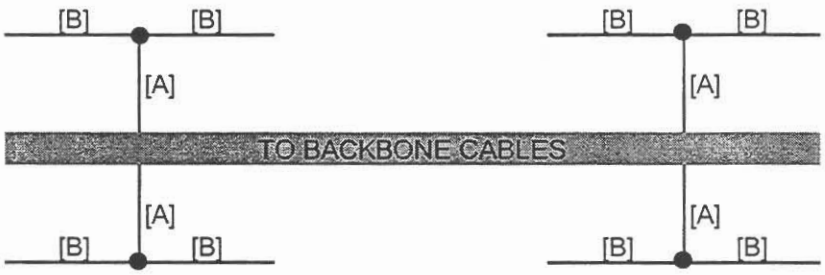
In all the nine distribution diagrams, the spans are labeled in ascending order – the span that supports the most number of spans is labeled "A" and so forth. Also, n*ote that the value of each span label is specific to that particular diagram.* For instance, in Distribution Style Three and above, a span labeled "A" support additional spans plus itself whereas in the Distribution Style One and the Distribution Style Two, the span labeled "A" supports only itself. In these two distribution styles, there are no span(s), other than span "A", coming out of the backbone cables to support.

Stick Map for Distribution Style Two



The Distribution Style Two is almost similar to the Distribution Style One with the exception that in this case instead of two spans coming out of the backbone cables, there are four spans. However, since all the spans support only itself, they are at the same demand level and are labeled "A".

Stick Map for Distribution Style Three



The Distribution Style Three is different from the two earlier styles. This time, the span labeled "A" supports two other spans including itself and the span labeled "B" support only itself. The distribution design is symmetrical within each side and between the two sides.

Stick Map for Distribution Style Four

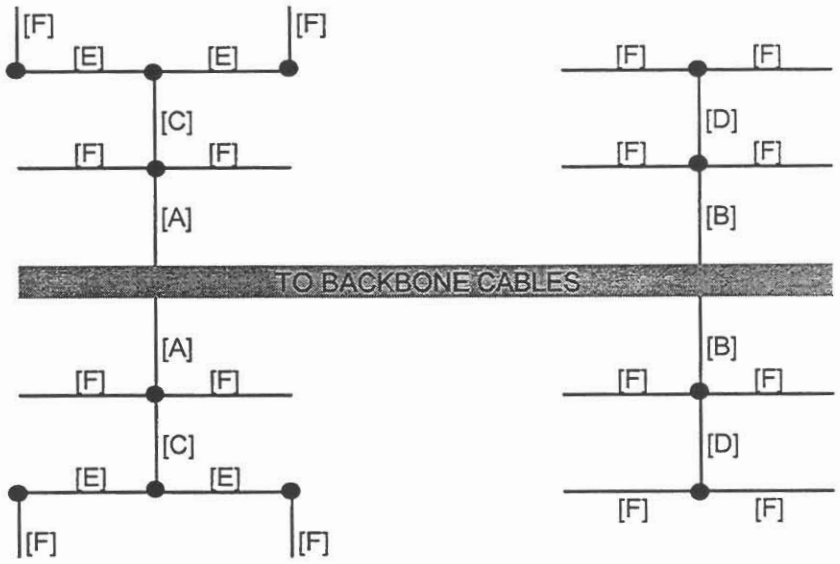


The Distribution Style Four shows an asymmetry of design between two sides and symmetry of design within each side. The first thing to notice here is that, unlike earlier distribution styles, the first span coming out of the backbone cables are labeled different.

On the right side of the backbone cables, the first span coming out of the backbone cables is labeled "B", whereas on the left side the cables the first span is labeled "A". The difference in labeling conveys different number of spans the span labeled "A" and the span labeled "B" supports – the span labeled "A" supports five spans plus itself and the span labeled "B" supports three spans plus itself.

Similarly, the span labeled "D" supports only itself and the span labeled "C" supports two spans plus itself.

Stick Map for Distribution Style Five

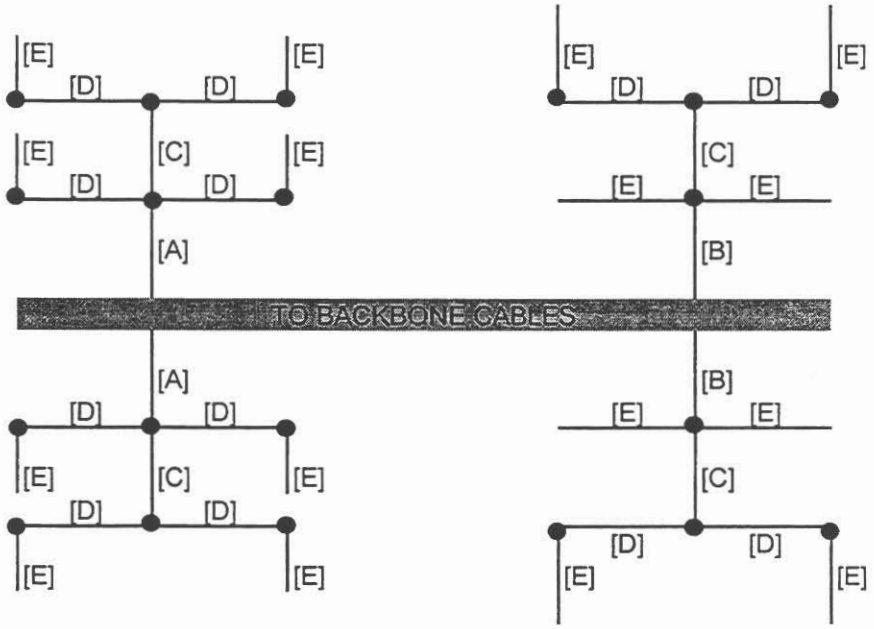


The Distribution Style Five is another example of a diagram that shows an asymmetry of design between the two sides of the backbone cables and symmetry of design within each side of the cable. In this case, the span labeled "A" supports seven spans plus itself and the span labeled "B" supports five spans plus itself. Whereas the span labeled "C" supports four spans plus itself, the span labeled "D" supports two spans plus itself.

Notice that unlike the case in the right side of the diagram, there is no span labeled "B" on the left side of the diagram. This is because, there is no span on the left side of the diagram that supports exactly five spans plus itself. In this diagram, the span labeled "B" is assigned to a span which supports exactly five spans plus itself.

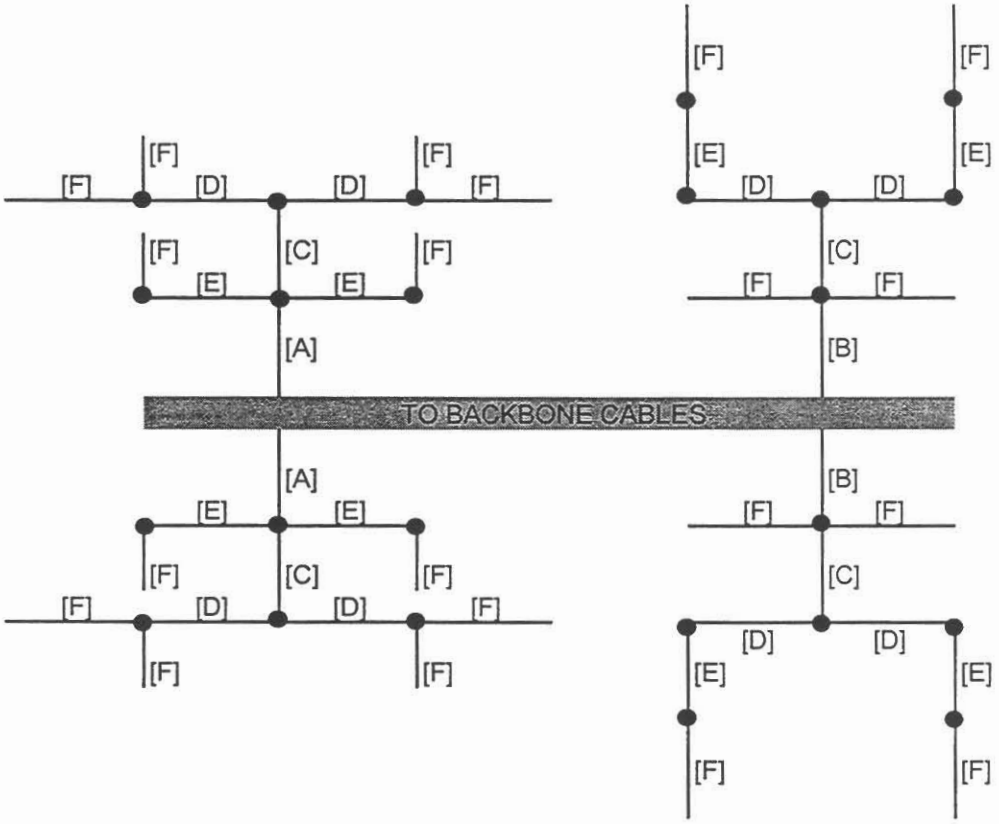
Again, note that the value of each span label is specific to that particular diagram. For instance, the span labeled "B' in the Distribution Style Five is supporting exactly five spans plus itself. However, the span labeled "B" in the Distribution Style Six is supporting exactly seven spans plus itself. The values of these two spans labeled, as "B", is different because each supports different number of spans.

Stick Map for Distribution Style Six



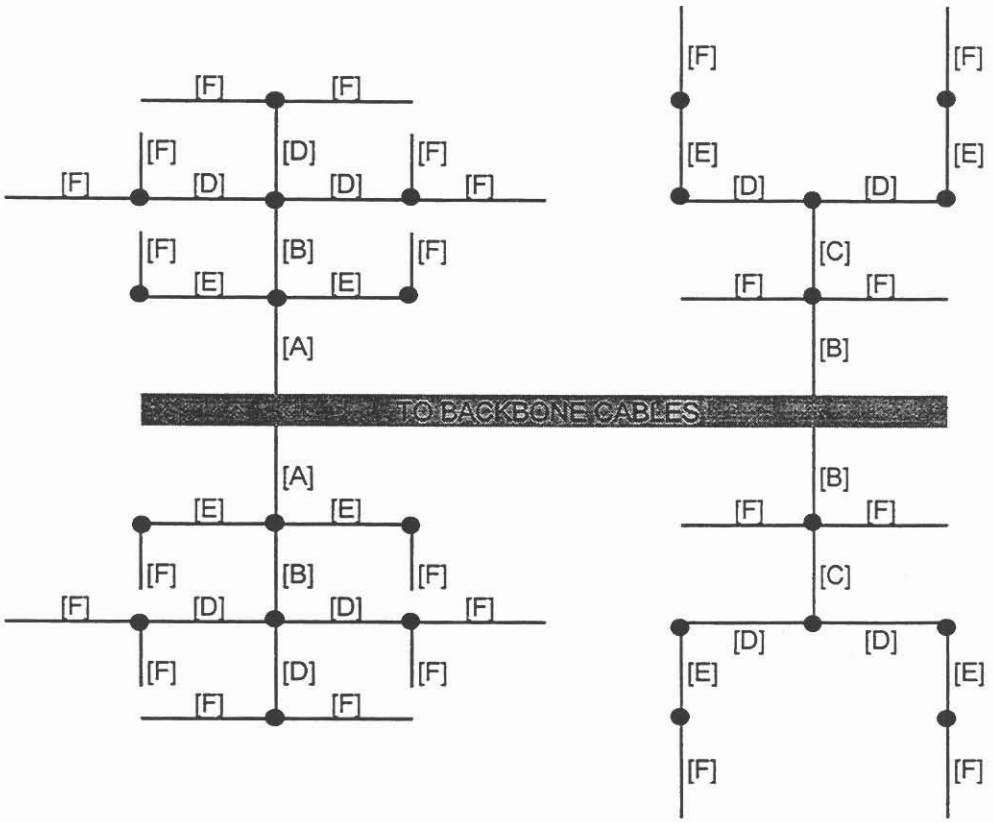
The Distribution Style Six is yet another example of a diagram that shows an asymmetry of design between two sides of the backbone cables and symmetry of design within each side. In this case, the span labeled "A" supports the highest number of spans, nine spans plus itself and the span labeled "E" supports the lowest number of spans, only itself. Whereas the span labeled "D" supports one other span plus itself, the span labeled "C" supports four other spans plus itself. Notice that unlike the case in the right side of the diagram, there is no span labeled "B" on the left side of the diagram. This is because, there is no span on the left side of the diagram that supports exactly seven spans plus itself. In this diagram, the span label "B" is assigned to a span that supports seven spans plus itself.

Stick Map for Distribution Style Seven



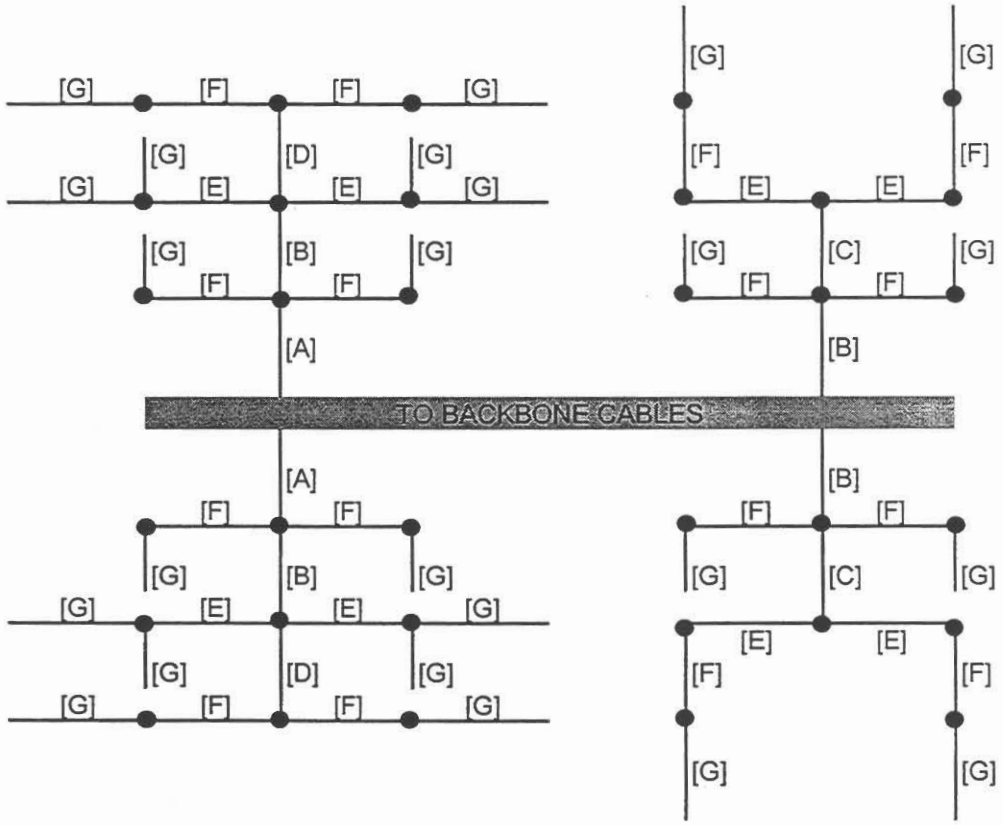
The Distribution Style Seven is yet another example of a diagram that shows an asymmetry of design between two sides of the backbone cables and symmetry of design within each side. As in all other cases, in this case too, the span labeled "A" supports the highest number of spans, eleven spans plus itself. Also, in this diagram, the span labeled "F" supports the lowest number of spans, only itself. Similarly, the span labeled "D" supports two other spans including itself and the span labeled "C" supports six other spans including itself. Notice that there is no span labeled "B" on the left side of the diagram. Again, this is the case because there is no span on the left side of the diagram that supports exactly nine spans plus itself.

Stick Map for Distribution Style Eight



The Distribution Style Eight has symmetry of design within each side the backbone cables and asymmetry of design between the two sides of the cables. In this diagram, the span labeled "A" supports thirteen spans plus itself and the span labeled "B" supports nine spans plus itself. The span labeled "F" supports the lowest number of spans, only itself.

Stick Map for Distribution Style Nine



The Distribution Style Nine has symmetry of design within each side of the backbone cables and asymmetry of design between the two sides of the cables. In this diagram, the span labeled "A" supports sixteen spans plus itself, and the span labeled "G" supports the lowest number of spans, only itself.

The nine distribution diagrams show several ways in which a user may design a distribution network. Depending on his/her data set, a user may use any one of the nine diagrams presented here, or any combination of them, to design his/her own network.

APPENDIX D: Loop Components

The Loop Components comprise of two parts: Feeder and Distribution. ICM deploys copper feeder cable for shorter loops and fiber feeder cable with Next Generation Digital Loop Carrier (NGDLC) for longer loops. The fiber feeder cables extend from the wire center out to a remote terminal. In the remote terminal, the NGDLC converts optical signals to electrical signals. The electrical signals then travel over copper distribution cables and drop wires to the end user's location.

The Figure 7 below identifies the feeder and distribution portions of loop and its components. The top drawing shows a copper loop with aerial distribution cable. The bottom drawing shows a fiber/copper loop that is all buried. A detail description of Distribution and Feeder components is provided following Figure 7.

The manner in which distribution and feeder cables are placed determines the support structure required. For instance, aerial cable is supported by poles, buried cable is placed in trenches, and underground cable requires conduit, manholes and pull boxes.

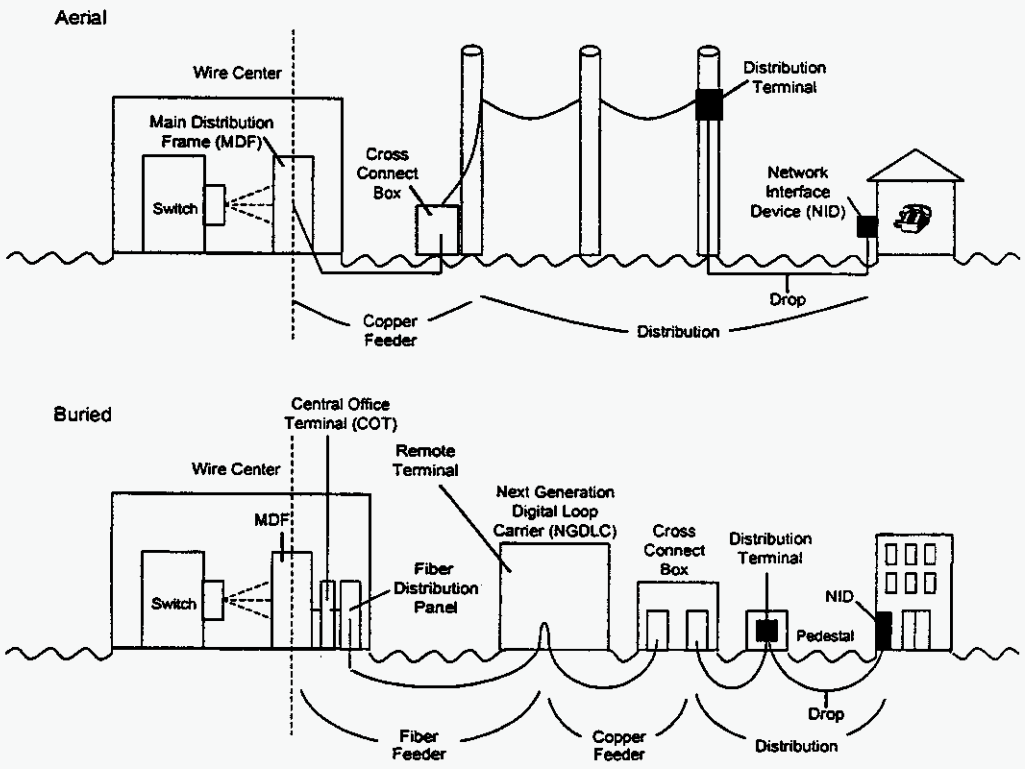


Figure 7: Aerial Copper and Buried Fiber/Copper Loops

Distribution Components

Distribution consists of the network components from the end-user's location up to, but not including, the cross connect box. The major components for distribution are listed below:

- **Network Interface Device (NID)** is the loop component that serves as the interface between the drop wire and the end user's inside wiring. The NID may be located inside or outside the end user's premises.
- **Drop Wire** for both residence and business is the loop component used to transport telephone service from the distribution terminal to the end user's NID. The drop wire is assumed to be buried copper cable.
- **Distribution Terminal** is the loop component that serves as the cross connect point between the distribution cable and the drop wire. The distribution terminal may be connected to

buried cable and located in a pedestal, or connected to aerial cable and mounted on a pole or building.

- **Distribution Cable** is the loop component used to transport telephone service from the cross connect box to the distribution terminal near the end-user's location. Distribution cable consists of copper cable and can be buried, underground, or aerial.

Feeder Components

Feeder consists of the network components from the cross connect box to the wire center. At the wire center, the copper feeder cables terminate on the Main Distribution Frame (MDF). The fiber optic feeder cables terminate on the Fiber Distribution Panel before extending to the Central Office Terminal (COT). The major components for feeder are listed below:

- **Cross Connect Box** is the loop component that serves as the connecting point between distribution and feeder.
- **Next Generation Digital Loop Carrier (NGDLC) or Pair Gain System** is the currently available technology required to accommodate fiber optic cable. When fiber feeder is used, the NGDLC is located in a Remote Terminal (RT) adjacent to the Cross-Connect Box and in the wire center in a COT.
- **Feeder Cable** is the loop component that transports telephone service via a copper cable from the Main Distribution Frame or DLC remote terminal to a cross connect box, or via a fiber cable from a DLC Central Office Terminal to a DLC Remote terminal. Feeder cable may be buried, underground, or aerial.
- **MDF Protector** is a protection device spliced to the end of the copper feeder cables and mounted on the Main Distribution Frame. It serves as a lightning suppresser to protect the switch.
- **Main Distribution Frame (MDF)** is a metallic frame used to mount the protectors spliced to the outside plant cables on one side and the connecting blocks, for terminating jumper cables to the switch and other central office equipment, on the other side. It serves as the interface between the loop and the switch.
- **Fiber Distribution Panel** is the termination point for fiber cable where the individual fibers are split apart and then extended to the COT.

For investment calculation purposes, the vertical side of the MDF and all of the MDF protectors are considered part of the loop. The horizontal side of the MDF is considered part of the switch.

This is a Blank Page



Integrated
Cost Model

Release 4.1

Model Methodology

Switch Module

(Book III of VII)

GTE Service Corporation

Copyright © GTE 1997 - 2000. All rights reserved.

This is a Blank Page

ICM MODEL METHODOLOGY PACKAGE

- Conceptual Framework — Book I
- Loop Module — Book II
- Switch Module — Book III
- Interoffice Transport Module — Book IV
- SS7 Module — Book V
- Expense Module — Book VI
- Mapping/Report Module — Book VII

This is a Blank Page

Table of Contents

ICM MODEL METHODOLOGY PACKAGE ..... a

SWITCH MODULE 1

Switching 1

Switching Components 2

Modeling Approach 2

Data Inputs 5

Wire Center Data 5

Switch Types 7

SCIS/COSTMOD 8

Switch Model Inputs 9

Composite Factors 9

Investment Adjustment Factor 10

Engineered, Furnished & Installed Factors 10

Power Investment 10

Test Equipment Investment 11

Model Processing 11

Outputs 11

APPENDIX A: Data Input Tables 13

Switch [SWITCH.DB] 14

EFI [XXEFI.DB] 15

Termination [XXSWINV.DB] 16

Usage [XXSWINV.DB] 17

Features [XXSWINV.DB] 18

Switch Investment [XXSWINV.DB] 19

Miscellaneous [XXSWINV.DB] 20

RTU FEES [XXSWINV.DB] 21

Switching Nodes [XXNODES.DB] 22

Features [FEATURE.DB] 24

Usage Parameters [USAGEP.DB and USGRATE.DB] 25
Processing Flow 26

APPENDIX B: Switched Features 27
Usage Features 27
Line Features 27
Custom Calling/Intelligent Network Features 27
CLASS Features 27
Centrex (CentraNet) Features 28
Centrex (CentraNet) Call Handling Features 29
Centrex (CentraNet) Attendant Features 29
Basic Business Group Features 30
ISDN PRI (Primary Rate Interface) 30
ISDN BRI (Basic Rate Interface) 30

SWITCH MODULE

The ICM Switch Module determines the relevant investments by component for each host switch and remote switch in GTE's network. Loaded switch investments, by component, are determined by adding power, test equipment, and EF&I (Engineered, Furnished, and Installed) to switch investment.

The expenses associated with each switching component are developed in the Expense Module. The Mapping/Report Module maps the components to elements and services and converts switch function investments to monthly costs, including busy hour investments for switch functions, such as call processing, to time of day investments.

Switching

Switching is the portion of the Public Switched Telephone Network (PSTN) that provides circuit switched connections between a calling party and a called party, or between a calling party and another switching network (for example, one belonging to an Interexchange Carrier). The switching network components extend from the main distribution frame (MDF) through the switch to the interoffice trunk termination. They are shown in Figure 1 below.

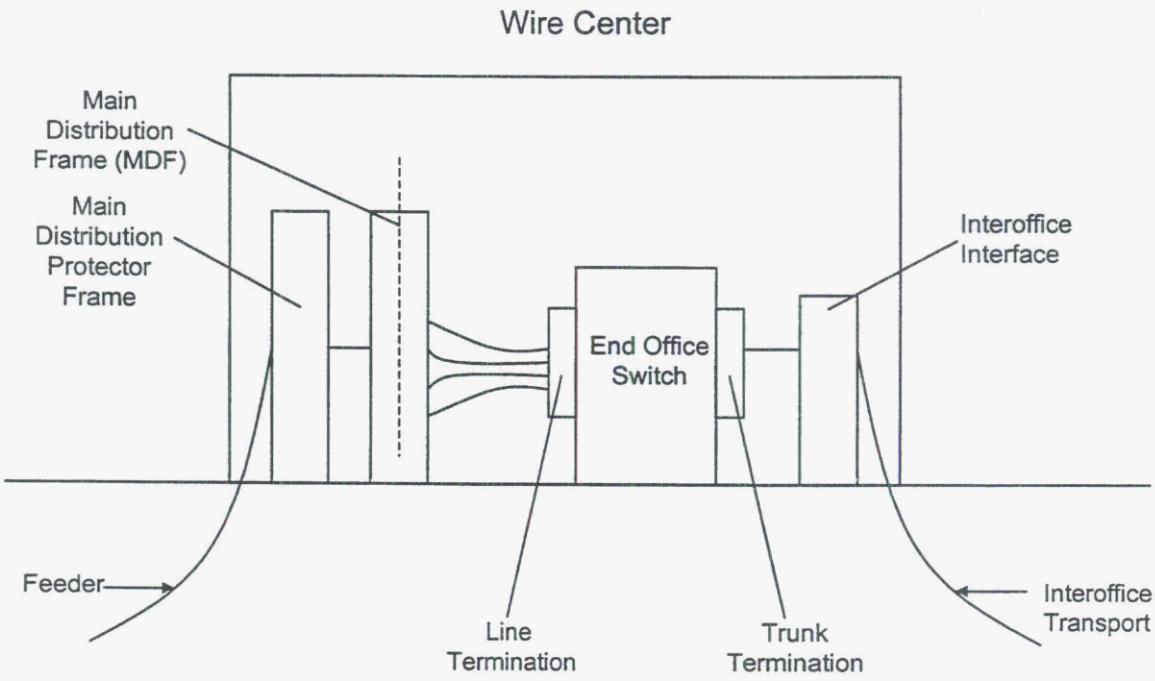


Figure 1: Switching Network Components

Calls that originate and terminate in the same switch are intra-office or line-to-line calls. Calls that originate in one switch and terminate in another are inter-

office calls. The switch connection is line-to-trunk in the originating office and trunk-to-line in the terminating office. Some inter-office calls are switched en route by a tandem switch, using a trunk-to-trunk connection.

Switching Components

The Switch Module estimates investments for the following components:

- Line terminations - Line side switch connection that connects individual loops to the switching components of GTE's network.
  - Analog, Coin, Integrated Services Digital Network Basic Rate Interface (ISDN BRI), Integrated Services Digital Network Primary Rate Interface (ISDN PRI)
- Trunk terminations - Trunk side switch connection that connects the switching components to other switches.
  - Digital DS-0
- Call setup and minutes of use (MOU) for the following call types:
  - Line to Line (intraoffice)
  - Line to Trunk (originating from end office)
  - Trunk to Line (terminating to end office)
  - Trunk to Trunk (tandem office or host/remote)
- Switched features - Features that enhance end user calling capability such as Custom Calling, CLASS, ISDN and CentraNet. See Appendix B for a complete list of features.

Modeling Approach

The Switch Module uses the outputs of the SCIS[1] and COSTMOD[2] models as the basis for developing the feature and function investments for each switch and remote in GTE's serving area. SCIS and COSTMOD calculate material investments for basic switching functions in each type of forward-looking switch deployed in GTE's network based upon office type, size and usage characteristics. The model results for material investments are then input into ICM.

Within ICM, busy hour call setup and MOU investments are converted to rate period-specific investments. One of two composite factors is applied to the

[1] As discussed more fully herein, the Switching Cost Information System (SCIS) was developed by Telcordia to model the investments for the features and functions of switching equipment.
[2] COSTMOD refers to GTE's proprietary cost models. The COSTMOD switching model was developed by GTE to provide switch investments for the GTD-5 technology. When the term COSTMOD is used in this document it is referring only to the COSTMOD switching model.

SCIS/COSTMOD outputs to determine loaded unit investments[3]. The factor depends on whether the output is a termination/usage investment or a switched feature investment. Both factors include loading for EF&I, power, and test investments. The factor for line or trunk terminations and usage also accounts for melded vendor pricing of initial switch purchases and additions.

Land and building expenses associated with switch investments are captured in the Expense Module. The switch right-to-use fees (RTU) are included in the SCIS/COSTMOD investment outputs.

Figure 2 is a flow chart depicting the Switch Module.

[3] Loaded unit investment includes the material vendor price and all GTE labor and minor materials required for installation.

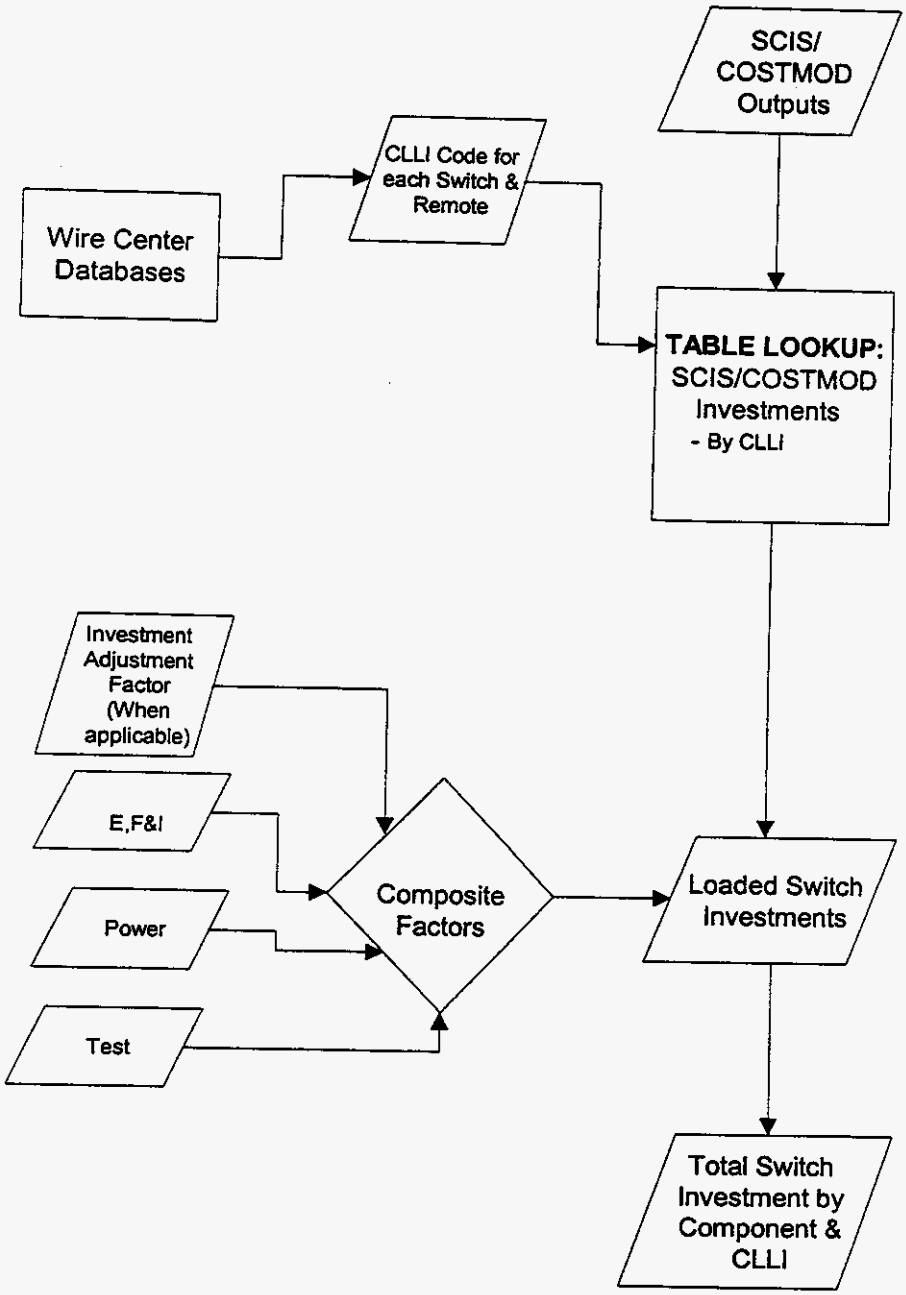


Figure 2: Switch Module Flow Chart

Data Inputs

The Switch Module uses data from the following sources to develop the switch investment primitives:

- Wire Center Databases
- SCIS/COSTMOD

The composite factors are themselves made up of factors that are developed outside of the model. The composite factors may include some or all of the following items:

- Investment Adjustment Factor
- Engineered, Furnished, and Installed (EF&I) Factors
- Power Investment
- Test Equipment Investment

The Switch Module inputs and their sources are described below.

Wire Center Data

Figure 3 below shows the different switch classes in GTE's network.

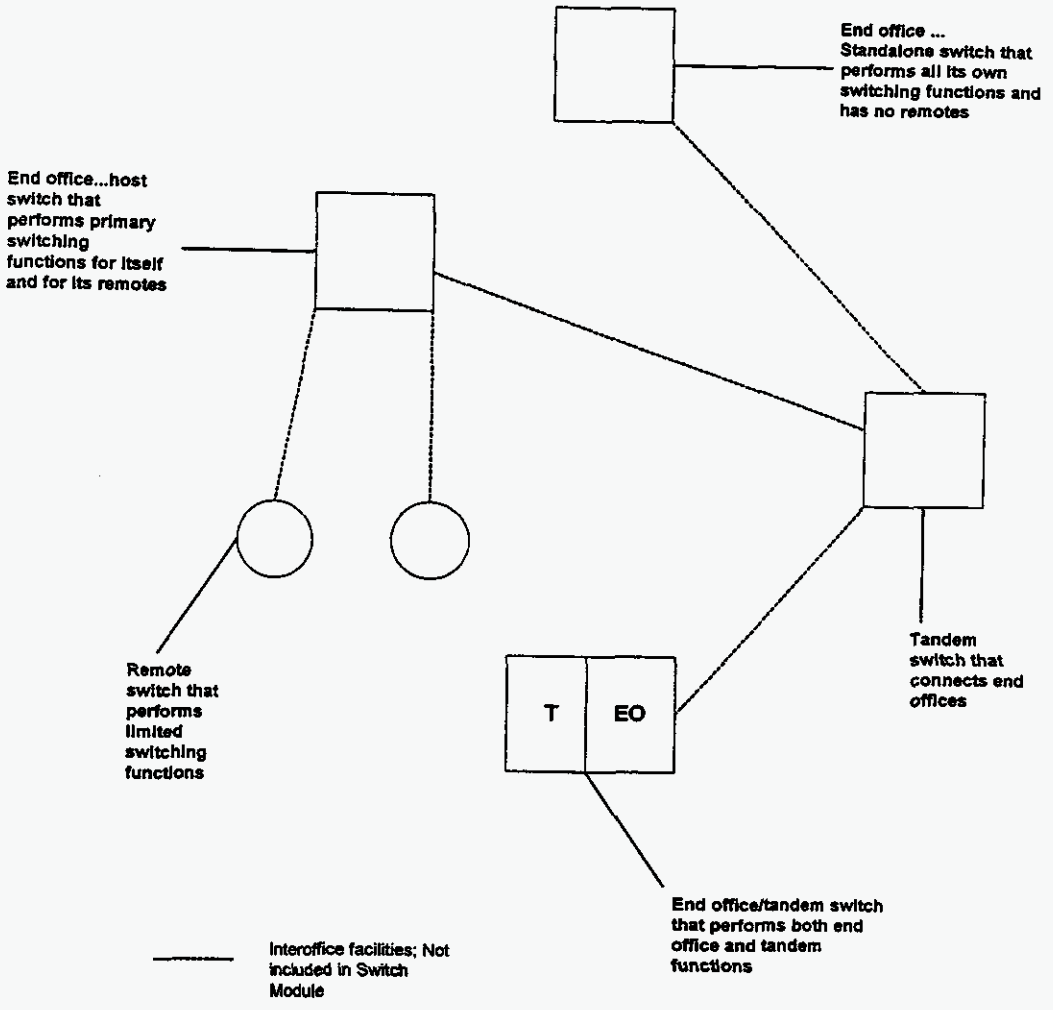


Figure 3: Switch Categories

The CLLI codes for GTE's switches in a particular state are derived from the NECA[4] 4 wire center database. The CLLI codes identify each switch by town, state, and switch class (host, remote, or tandem). The location of each switch in latitude and longitude is obtained from the CLONES[5] database.

GTE provides the following information for each switch:

- Switch vendor and technology type
- Homing arrangements that identify the CLLI code of the host end office for each remote

[4] National Exchange Carrier Association
[5] Central Location On-Line Entry System

- Homing arrangements that identify the CLLI code of the tandem for each end office

Switch Types

The Switch Module uses four digital switch types (5ESS, DMS-10, DMS-100, and GTD-5) that represent the forward-looking technology that GTE will use in its network. The switch vendors, models, and classes that may be represented in ICM by these switch types are shown below:

Switch Type	Class
Lucent 5ESS (end office)	5
Lucent 5ESS (tandem/end office)	4/5
Lucent 5ESS (tandem)	4
Lucent 5ESS Remote	5
Nortel DMS-10 (end office)	5
Nortel DMS-10 Remote	5
Nortel DMS-100 (end office)	5
Nortel DMS-100/200 (tandem/end office)	4/5
Nortel DMS-100 Remote	5
Nortel DMS-200 (tandem)	4
Lucent/AGCS GTD-5 (end office)	5
Lucent/AGCS GTD-5 (tandem/end office)	4/5
Lucent/AGCS GTD-5 Remote	5

GTE includes these switch types in its existing network and will continue to support them in the future.

GTE currently has switches in its network that are not one of the four types included in the Switch Module. If the wire center database indicates that a switch other than those listed above is installed at a particular location, one of the above is substituted in its place.

In addition, in some states, GTE has replaced electro-mechanical switches with pair gain devices. These pair gain devices retain a CLLI code because they have replaced obsolete switches. When they appear in the wire center database, the code REMX_(host switch type) is placed in the substitution column of the XXNODES.DB table.

SCIS/COSTMOD

SCIS and COSTMOD are engineering-oriented, PC-based models that are used to produce investments for features and functions of the switching equipment in GTE's telecommunications network. SCIS determines switching investments for the equipment GTE purchases from Nortel and Lucent Technologies. COSTMOD produces switching investments for the Lucent/AGCS switching technology. Both SCIS and COSTMOD have gained regulatory acceptance within the United States. Based on the switch-specific engineering rules provided by the manufacturer, office size, and traffic and demand inputs, these switching cost models calculate the material unit investments for features and functions by switch type.

This engineering orientation requires a set of inputs for each office modeled. Some inputs are technology-specific (5ESS, DMS-100/200, DMS-10, or GTD-5) because each technology is different in its design. However, the basic inputs are similar and include the following:

- Type of office (host, standalone, remote, tandem)
- Number of lines (including integrated DLC lines)
- Line traffic characteristics (for example, CCS (centum, or 100, call seconds) per line and the line concentration ratio)
- Processor utilization
- Number of trunks (local and tandem)
- Trunk traffic characteristics (for example, calls and CCS per trunk)
- Traffic characteristics for switched features
- Switch vendor discounts
- Average fill

The outputs are investments by feature and function for central office and remote equipment.

The following total and unit investments are outputs of the switching models:

- Getting started investment per millisecond (related to processing time for call-setup)
- Investment per line termination
- Investment per line CCS
- Investment per call
- Investment per trunk CCS
- Investment per tandem trunk CCS

- Investment per switched feature (based on the above outputs for basic switch functions)

Switched feature investments (for example, call waiting) are based on vendor engineering rules, user inputs of feature characteristics, and modeled office investments for switching functions. Switch function investments capture the costs of the switch, such as processor real time, used to provide switched calls and features. Network characteristics and user demand are then used to determine the specific switching resources used by a particular feature. These calculations are carried out within the models, and investment reports are generated for each feature.

SCIS and COSTMOD are run in average and marginal/capacity modes to provide investment inputs for ICM. Marginal/capacity investments are used to estimate volume sensitive (VS) investments. The difference between average and marginal/capacity investments is used to estimate volume insensitive (VIS) investments. Average investments (the equivalent of the summation of VS and VIS) are used to produce average costs. For additional information, see the SCIS and COSTMOD output reports in GTE's filing package.

Switch Model Inputs

To develop the inputs for ICM, engineering data (lines, trunks, call attempts, CCS) was retrieved from GTE's engineering systems. These systems are:

- Traffic Data Collection System
- Trunk Traffic Estimator
- Traffic Sensitive Forecast

Each switch was then modeled through SCIS and CostMod to produce unique investments for each CLLI based on site specific data.

Composite Factors

Composite factors are developed within ICM to convert switch material unit investments to loaded investments. The composite factors applied to usage, and line and trunk investments include investment adjustment; engineered, furnished, and installed; power; and test factors. The factors applied to switched features contain the same components with the exception of the investment adjustment factor. Each of the factor components, described below, is developed in studies external to ICM. To review the detailed calculations, refer to the *ICM Module Support Documentation.*

Investment Adjustment Factor

To ensure that switch investments are forward-looking, switch list prices are discounted to reflect GTE vendor contracts and quotes. Vendors' quotes typically include separate prices for new switch placements and for additions to existing switches. The ICM switch investment outputs for usage and line and trunk terminations reflect a meld of the two discounts.

Since discounts for new switch purchases are input into SCIS and COSTMOD, these models' outputs (usage and line- and trunk terminations) are adjusted within ICM using the Investment Adjustment Factor to reflect the meld of new and additions pricing. The Investment Adjustment Factor does not apply to switched feature investments. The switched feature outputs reflect only the new switch discount price.

Engineered, Furnished & Installed Factors

Two Engineered, Furnished and Installed (EF&I) factors are developed to include labor investments for engineering and installation and minor materials. They are listed below:

EF&I - I: Includes installation costs that occur in the time between delivery of equipment from the vendor and when the switch is turned over to the local operating forces.

EF&I - EF: Includes costs for GTE engineering and furnishing, local installation, acceptance, and supply loadings.

Both factors vary by switch type and line size category. Since labor rates and minor material and supply loadings differ by state, state-specific EF&I factors are developed for each state.

Power Investment

All of the equipment, rectifiers, and batteries, required to provide DC power and backup power to the switch are included in power investment. Power investment values are developed using engineering and vendor guidelines and account for both labor and material costs.

Although the power investments do not vary by switch type, they do vary by switch and remote size as shown below:

- Small (up to 5,000 lines)
- Medium (8,300 lines)
- Large (13,300 lines)
- Very Large (29,200 lines)
- Extra Large (60,000 lines)
- Small Remote (up to 2,600 lines)
- Large Remote (3,750 lines)

Unique power investments are developed for each state to reflect state-specific labor costs.

Test Equipment Investment

The test equipment investment includes miscellaneous monitoring and administrative test equipment. It is determined by using GTE's engineering practices and vendor contracts. Test material investment is constant for all host switch types and sizes. Although test equipment material investment does not vary by state, however, after loading state-specific EF&I factors to test equipment material the cost of the test equipment investment will vary by state.

Model Processing

The Switch Module determines the loaded unit investments for each GTE host and remote switch in a state. SCIS/COSTMOD switch investments for line and trunk terminations are pulled from the switching investment table (XXSWINV.DB) based on the switch CLLI.

The investment for usage and switched features for hosts is also pulled from the (XXSWINV.DB) table. SCIS/COSTMOD outputs are inputs in the (XXSWINV.DB) table. For remote switches, the usage and switched feature investments for the associated host switch are used.

When a pair gain device has been substituted for a switch (REMX in the Substitution column of the XXNODES.DB table), the Switch Module uses the appropriate digital loop carrier (DLC) investments from the Outside Plant and Material table[6] in place of the line and trunk terminations. Usage and switched feature investments are calculated, like remotes, based on the host end office CLLI.

Busy hour usage related investments for remote switching, end office switching, tandem switching, and automatic message accounting (AMA, i.e., Billing Expenses) are converted to an investment per call setup and minute of use (MOU) by rate period, using GTE's usage methodology.

Once the switch investments are determined, the appropriate composite factors are applied to reflect investment adjustments (when applicable), EF&I, power, and test loadings.

The switch investment components are then passed to the Mapping/Report Module where costs are calculated for basic network functions, unbundled network elements, and services.

Outputs

The Switch Module outputs are used to develop monthly costs for the following:

[6] Digital loop carrier investments are used in the Mapping/Report Module for calculating loop investment.

- Line Terminations
- Trunk Terminations
- Ports
- Switched Features
- End Office Switching
  - Average Minutes of Use
  - Originating Call Setup, Minutes of Use
  - Terminating Call Setup, Minutes of Use
  - Intraoffice Call Setup, Minutes of Use
- Switching - AMA Recording (Automatic Message Accounting)
- Tandem Switching
  - Average Minutes of Use
  - Minutes of Use, Call Setup.

APPENDIX A: Data Input Tables

The data input tables are listed below. The tables can be viewed and printed under the menu option **Tables/Switch.** The name of each database file is listed; often the first two letters indicate state (that is, xxexbase.db). A user may modify data in all of the tables listed below. Detailed descriptions of each of the tables are provided in the following section.

Following the description of data input tables, a "bubble chart" for Switching is provided under the heading 'Processing Flow'. The chart contains a graphic representation of the database files that are inputs to and outputs of the Switch Module.

Switch

- Switching
  - Switch (SWITCH.DB)
  - EFI (XXEFI.DB)
  - Termination (XXSWINV.DB)
  - Usage (XXSWINV.DB)
  - Features (XXSWINV.DB)
  - Switch Investment (XXSWINV.DB)
  - Miscellaneous (Power, Test equipment, and Investment Adjustment Factor) (XXSWINV.DB)
  - RTU Fees (XXSWINV.DB)
- Nodes
  - Switching Nodes (XXNODES.DB)
- Features
  - Features (FEATURES.DB)
- Usage Parameters (USAGEP.DB and USGRATE.DB)
- Processing Flow
  - Switching

Switch [SWITCH.DB]

Purpose: Lists the type of switch used and whether the switch is a Host or a Remote. The switches used represent forward-looking technology that GTE will use in its network.

COLUMN HEADINGS	DESCRIPTION
SWITCH TYPE	One of four switch types (5ESS, DMS-10, DMS-100, or GTD-5), plus REM to designate remote switches. The following specific switch types are utilized by GTE: • Lucent 5ESS (end office and remote) • Nortel DMS-10 (end office and remote) • Nortel DMS-100 (end office and remote) • Lucent/AGCS GTD-5 (end office and remote)
ID	Designates switch type for mapping purposes
HOST/REM	H or R indicates whether the switch is a host switch located in the end office, or a remote. D indicates a Digital Pair Gain serving as a wire center.

EFI [XXEFI.DB]

Purpose: Lists the EF&I (engineered, furnished and installed) factors, the Investment Adjustment Factor, the Power Investment, the Test Investment, and the Loop Test Investment for host and remote switches by Switch Type and Size.

COLUMN HEADINGS	DESCRIPTION
SWITCH TYPE	Type of host or remote switch
SIZE	Line size category.
EFI - I	GTE Installation: Includes installation occurring between delivery of equipment from the vendor until the switch is turned over to the local operating forces.
EFI - EF	GTE Engineering and Furnishing: Includes GTE engineering and furnishings, local installation, acceptance and supply loadings.
IAF	Investment Adjustment Factor: A factor applied to initial switch cost to adjust for growth switch cost. The result is weighted investment for switch.
POWER INV	Power Investment: Loaded cost of Central Office DC power equipment
TEST INV	Test Investment: Material cost of monitor and test equipment
LOOP TEST	Loop Test Investment: Material cost of 4 TEL Loop Test Equipment

Termination [XXSWINV.DB]

Purpose: Lists termination investments by CLLI

COLUMN HEADINGS	DESCRIPTION
CLLI	Common Language Location Identifier (CLLI) code of the switch
INVESTMENT TYPE	Code identifying the switch investment type T0001 Line Termination Analog T0002 Trunk Termination T0003 Line Termination Digital T0004 Analog Trunk Termination T0005 Coin Line Termination T0006 BRI T0007 PRI B T0008 PRI D
CATEGORY	Code representing type of investment. F – Feature L – Line T – Termination U – Usage M – Miscellaneous R – RTU Fees
AVERAGE INVEST	SCIS/COSTMOD average investment in dollars
VS INVEST	SCIS/COSTMOD marginal investment in dollars

Usage [XXSWINV.DB]

Purpose: Lists usage investments by CLLI.

COLUMN HEADINGS	DESCRIPTION
CLLI	Common Language Location Identifier (CLLI) code for switch
INVESTMENT TYPE	Code identifying switch usage investment type. U0000 AMA U0001 Line – Line Call Setup U0002 Line - Line Minutes U0003 Line – Trunk Call Setup U0004 Line – Trunk Minutes U0005 Trunk – Line Call Setup U0006 Trunk – Line Minutes U0007 Trunk – Trunk Call Setup U0008 Trunk – Trunk Minutes
CATEGORY	Code representing type of investment. F – Feature L – Line T – Termination U – Usage M – Miscellaneous R – RTU Fees
AVERAGE INVEST	SCIS/COSTMOD average investment in dollars
VS INVEST	SCIS/COSTMOD marginal investment in dollars

Features [XXSWINV.DB]

Purpose: Lists feature investments and numbers by CLLI.

COLUMN HEADINGS	DESCRIPTION
CLLI	Common Language Location Identifier (CLLI) code for switch
INVESTMENT TYPE	Code assigned to switch features included in ICM.
CATEGORY	Code representing type of investment. F – Feature L – Line T – Termination U – Usage M – Miscellaneous R – RTU Fees
AVERAGE INVEST	SCIS/COSTMOD average investment in dollars
VS INVEST	SCIS/COSTMOD marginal investment in dollars

Switch Investment [XXSWINV.DB]

Purpose: Lists investment types by CLLI

COLUMN HEADINGS	DESCRIPTION
CLLI	Common Language Location Identifier (CLLI) code for switch
INVESTMENT TYPE	L0001 – Indicates switch investment
CATEGORY	Code representing type of investment. F – Feature L – Line T – Termination U – Usage M – Miscellaneous R – RTU Fees
AVERAGE INVEST	SCIS/COSTMOD average investment in dollars
VS INVEST	SCIS/COSTMOD marginal investment in dollars

Miscellaneous [XXSWINV.DB]

Purpose: Lists miscellaneous investments by CLLI.

COLUMN HEADINGS	DESCRIPTION
CLLI	Common Language Location Identifier (CLLI) code for switch
INVESTMENT TYPE	Indicates type of adjustment applied to switch investments. M0001 Power M0002 Test Equipment M0003 Investment Adjustment Factor
CATEGORY	Code representing type of investment. F – Feature L – Line T – Termination U – Usage M – Miscellaneous R – RTU Fees
AVERAGE INVEST	Amount of investment (power and test) or factor (Investment Adjusted Factor).

RTU FEES [XXSWINV.DB]

Purpose: Lists Right To Use Fees by CLLI.

COLUMN HEADINGS	DESCRIPTION
CLLI	Common Language Location Identifier (CLLI) code for switch
INVESTMENT TYPE	Code assigned to feature RTU fees in ICM.
CATEGORY	Code representing type of investment. F – Feature L – Line T – Termination U – Usage M – Miscellaneous R – RTU Fees
AVERAGE INVEST	SCIS/CostMod Investment in dollars.

Switching Nodes [XXNODES.DB]

Purpose: A comprehensive listing of wire center data by CLLI

COLUMN HEADINGS	DESCRIPTION
CLLI	Common Language Location Identifier (CLLI) code of the switch
HOST CLLI	CLLI code of the Host switch
SWITCH TYPE	Type of switch, such as DMS-100 or 5ESS
CLASS	Class 4 (Tandem only), Class 4/5 (Tandem part of Tandem/End Office), or Class 5 (End Office only or End Office part of Tandem/End Office))
HML	Indicates density of area (populated after ICM processing) H = > 1,000 lines per square mile Buried feeder cable is trenched. Percent hand dig, concrete removal and boring are applied. 50 < M <= 1,000 lines per square mile Buried feeder cable is trenched. Percent hand dig, concrete removal and boring are applied. 0 <= L < = 50 lines per square mile Buried feeder can be plowed. Percent hand dig, concrete removal and boring are not applied.
LATITUDE	Latitude of the Wire Center
LONGITUDE	Longitude of the Wire Center
TOTAL LINES	Total number of lines served by the demand units assigned to the wire center (populated by ICM after processing)
NON SWITCHED LINES	Non-Switched lines for the Wire Center
DEMAND POINTS	Number of records in the Demand Table with demand > 0
LINE TERMS	Total number of lines served by copper feeder in the demand units assigned to the wire center (populated by ICM after processing)
BRI	Number of Basic Rate Interface ISDN lines for the wire center from GTE ARMIS data. Used to create a weighted average cost for ISDN UNEs.
PRI	Number of Primary Rate Interface ISDN lines per wire center from GTE ARMIS data. Used to create a weighted average cost for ISDN UNEs.
PBX	Private Branch Exchange (PBX): Number of PBX Lines

COLUMN HEADINGS	DESCRIPTION
KEY	Number of Key Lines
CENTRANET	Number of Centranet Lines
COIN	Number of Coin Lines
B1	Number of Business Lines
ROAD FEET	Total number of road feet in the demand points that are assigned to the wire center
ROAD FEET RATIO	Total road feet with demand divided by total road feet
BEDROCK	Measure of BedRock depth in inches
WATER TABLE	Measure of Water Table depth in inches
CLUSTER COUNT 12 KFt	The number of clusters for the Wire Center when 12 Kft loop limit is used. (Updated by ICM).
DISTANCE ADJ	The amount subtracted from the maximum loop length to determine the maximum feeder length in feet. (Height + Width) divided by 2
WIDTH DEMAND PT	The distance in feet of one two-hundredth of a degree of latitude
HEIGHT DEMAND PT	The distance in feet of one two-hundredth of a degree of longitude
DU	Distribution capacity copper utilization (fill) factor
FU	Feeder capacity fiber utilization (fill) factor
F AERIAL PCT	Portion of fiber facilities that is aerial
F BURIED PCT	Portion of fiber facilities that is buried
F UNDGND PCT	Portion of fiber facilities that is Underground
C AERIAL PCT	Portion of copper facilities that is aerial
C BURIED PCT	Portion of copper facilities that is buried
C UNDGND PCT	Portion of copper facilities that is underground
TIER B	Designates category of labor activity rates from the Outside Plant Placement Table(XXLABR.DB) to be used. TRUE = Tier B rates, FALSE = Tier A rates.

Features [FEATURE.DB]

Purpose: Lists the feature name and number assigned in the model

COLUMN HEADINGS	DESCRIPTION
FEATURE	Alphanumeric code for each switch feature (e.g., F0001 for Three Way Calling)
CATEGORY	Code representing type of investment. F – Feature L – Line T – Termination U – Usage M – Miscellaneous
DESCRIPTION	Brief description of the feature (for example, Automatic Callback)
GROUP	Indicates whether the feature applies to an individual line or a business line group.

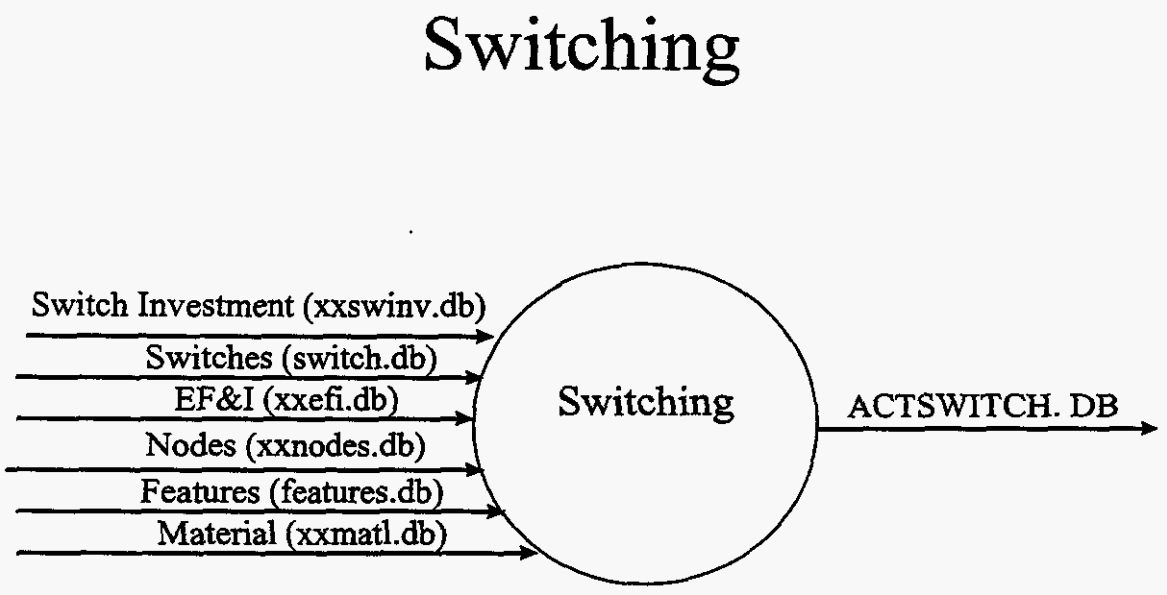
Usage Parameters [USAGEP.DB and USGRATE.DB]

Purpose: Lists switch parameters such as CCS per Trunk, Call Completion Ratio, SSP Setup Time, and Ringing Time

COLUMN HEADINGS	DESCRIPTION
DESCRIPTION	Description of Parameters
DAY	Parameters related to the usage in the daytime billing period (typically between 8:00 a.m. to 5:00 p.m.)
EVENING	Parameters related to the usage in the evening billing period (typically between 5:01 p.m. to 11:00 p.m.)
NIGHT	Parameters related to the usage in the night billing period (typically between 11:01 p.m. to 8:00 a.m.)
PEAK	Peak usage part of the day
OFF PEAK	Off-peak usage part of the day
24 HOUR	Parameters for costing usage sensitive services during anytime of the day

Processing Flow

The Switching bubble-chart provides a graphic view of the database files that are input to and output of the Switch Module.



APPENDIX B: Switched Features

Usage Features

- AMA Toll Call
- Line-to-Line Call Setup
- Line-to-Line Minute of Use
- Line-to-Trunk Call Setup
- Line-to-Trunk Minute of Use
- Trunk-to-Line Call Setup
- Trunk-to-Line Minute of Use
- Trunk-to-Trunk Call Setup
- Trunk-to-Trunk Minute of Use

Line Features

- Remote Call Forwarding
- Direct Inward Dialing - Directory Number
- Warm Line

Custom Calling/Intelligent Network Features

- 3-Way Calling
- Call Forwarding Variable
- Changeable Speed Calling 1-Digit
- Changeable Speed Calling 2-Digits
- In Contact
- Call Waiting
- Cancel Call Waiting
- Dual-Tone Multi-frequency (DTMF) Dialing
- Teen Service (Restricted Distinctive Alerting)

CLASS Features

- Automatic Callback
- Automatic Recall
- Anonymous Call Rejection
- Calling Number Delivery
- Calling Number Delivery Blocking
- Selective Call Waiting
- Caller ID Name and Number
- Distinctive Ringing / Call Waiting
- Customer Originated Trace
- Selective Call Acceptance
- Selective Call Forwarding
- Selective Call Rejection
- Calling Name Delivery
- Call Waiting ID

Centrex (CentraNet) Features

- Automatic Alternative Routing
- Call Forwarding Variable
- Call Forwarding Incoming Only
- Call Forwarding Within Group Only
- Call Forwarding Busy Line
- Call Forwarding Don't Answer All Calls
- Call Waiting Originating
- Call Waiting Terminating
- Call Flip-Flop
- Three Way Calling
- Call Transfer Individual All Calls
- Add-On Conversation Hold Incoming Only
- Speed Calling Individual 1-Digit
- Speed Calling Individual 2-Digits
- Customer Dialed Account Recording
- Business Set Group Intercom All Calls
- Last Number Redial
- Authorization Code Immediate Dialing
- Dial Call Waiting
- Loudspeaker Paging
- On-Hook Queuing for Outgoing Trunks
- Off-Hook Queuing for Outgoing Trunks
- Voice / Data Protection
- Authorization Codes for AFR
- Account Codes for AFR
- Code Restriction and Diversion
- Code Calling
- Tie Facility Access
- OUTWATS-Simulated Facility Group
- Direct Connect
- Distinctive Alerting / Call Waiting Indicator
- Call Hold
- Semi-Restricted (Originating and Terminating)
- Fully Restricted (Originating and Terminating)
- Toll Restricted Service
- Call Pick-up
- Directed Call Pick-up with Barge In
- Directed Call Pick-up without Barge In
- Special Intercept Announcements
- Conference Calling 6-Way
- Station Message Detail Recording to RAO
- Station Message Detail Recording to Premises
- Cancel Call Waiting
- Call Forwarding Enhancements (Multipath)
- Executive Busy Override
- Direct Inward System Access
- Recorded Telephone Dictation
- Business Set Call Pickup
- Business Set Access to Paging
- Auth Code for Message Detail Record
- Call Forwarding All (Fixed)
- Call Park
- Dual-Tone Multifrequency (DTMF) Dial 2
- Meet-Me Conference
- Music on Hold
- Business Set Call Back Queuing
- 800 Service-Simulated Facility Group
- Basic Business Set

Centrex (CentraNet) Call Handling Features

- Circular Hunting
- Preferential Multi-line Hunting
- Uniform Call Distribution
- Stop Hunt Key
- Make Busy Key
- Queuing for Multi-line Hunt Groups
- Automatic Route Selection
- Facility Restriction Level
- Expensive Route Warning Tone
- Manual / Time of Day Routing Control
- Foreign Exchange Facilities
- Deluxe Auto Route Selection

Centrex (CentraNet) Attendant Features

- Fixed Night Service Key
- Attendant Camp-on (Non-data Link)
- Attendant Busy Line Verification
- Attendant Control of Facilities
- Locked Loop Operation
- MDC Attendant Console (per A/G)
- Attendant Autodial
- Attendant Console Test
- Attendant Lockout
- Attendant Secretary
- Attendant Flexible Console Alerting
- Attendant Console Act/Deact of CFU/CFI
- Attendant Interposition Transfer
- Attendant on Incoming Lines
- Fixed Night Service – Call Forwarding
- Attendant Conference
- Attendant Recall from Satellite
- Attendant Position Busy
- Two-Way Splitting
- Attendant Call Park
- Attendant Speed Calling
- Attendant Delayed Operation
- Attendant Multiple Listed Directory Numbers
- Attendant Wildcard Key
- Attendant VFG Trunk Group Busy on Attendant Console
- Attendant Display of Queued Calls by IC Key
- Attendant Recall

Basic Business Group Features

- Basic Business Group
- Basic Business Group Direct Outward Dial
- Basic Business Group Station-to-Station Inter.
- Basic Business Group Auto ID Outward Dialing
- Privacy Release
- Six-Port Conference
- Basic Business Group Speed Calling Shared
- Basic Business Group Speed Calling 2 Shared
- Basic Business Group Automatic Callback
- Basic Business Group Direct Inward Dialing
- Display Calling Number

ISDN PRI (Primary Rate Interface)

- Primary Rate Interface
- Circuit Switched Voice/Data for PRI
- Call by Call Access
- Calling Number Delivery to PRI
- Non-Facility Associated Signaling
- Switched Fractional DS1 Originating
- Switched Fractional DS1 Terminating
- PRI D-Channel Backup
- PRI B-Channel

ISDN BRI (Basic Rate Interface)

- Attendant Busy Verification of Lines/Trunks
- Attendants Call Thru Test
- Shared Call Appearances of a DN
- Bridged Call Exclusion
- Key Systems Coverage for Analog line
- Queuing for ISDN Attendants w/c WI
- Attendants Control of Voice Terminals
- Attendant Night Service (Fixed/Flexible)
- Emergency Access to Attendant
- Attendant Direct Trunk Group Selection
- Attendant Emergency Override
- Auto Dropback to Attendant
- Attendant Original Permission Display
- Attendant Timed Reminder
- Attendant Trunk Identification
- Aggregate Work Time/Number of Calls Handled
- Total Number of Calls Handled Display
- Attendant Traffic
- Attendant Number of Calls on Queue
- Packet Switching IEO On Demand B Channel
- Circuit Switched Voice
- Basic Circuit Switched Data
- Packet Switching IAO D Channel
- X.25 Hunt Group
- Outgoing Calling Line ID
- Attendant Power Failure Transfer
- EDS Calling Name Display
- Attendant Camp-On
- Attendant Uniform Call Distribution
- Call Forwarding Variable

- ISAT Trunk Queuing
- Attendant Trunk Group Indicators
- Call Pick Up
- Attendant Direct Station Selection
- Multiline Hunt Group
- Attendant Position Busy
- Call Hold
- Toll Restricted Service
- Intercom Functions
- Priority Calling Incoming Only
- X.25 Closed User Groups
- X.25 Fast Select Acceptance
- X.25 Reverse Charge
- X.25 Permanent Virtual Call Service
- Code Calling-Answer
- Trunk Answer Any Station
- Bridging
- ISDN Time and Date Display
- Transit Delay Selection & Ind.
- ISDN Facility Restriction Level
- Flow Control Parameter Negotiation
- Outgoing Calls Barred
- Attendant Control of Facilities
- Attendant ID on Incoming Calls
- Business Group Auto Callback
- Attendant Conference
- Circular Hunting
- Attendant Call Hold
- Attendant Call Splitting
- Attendant Through Dialing
- Terminal Management
- Multiple Directory Number Button
- X.25 Fast Select
- X.25 One-way Outgoing Logical Channel
- X.25 Reverse Charge Acceptance
- Direct Connect
- Attendant Serial Call
- Delayed and Abbreviated Ringing
- Intercom Alerting
- Feature Button Inspect
- Inspect for ISDN Terminals
- ISDN Display for Ringing Call Appear
- Incoming Calls Barred
- Throughput Class Negotiation

This is a Blank Page



Integrated
Cost Model

Release 4.1

Model Methodology

Interoffice Transport Module

(Book IV of VII)

GTE Service Corporation

Copyright © GTE 1997 - 2000. All rights reserved.

This is a Blank Page

ICM MODEL METHODOLOGY PACKAGE

- Conceptual Framework — Book I
- Loop Module — Book II
- Switch Module — Book III
- Interoffice Transport Module — Book IV
- SS7 Module — Book V
- Expense Module — Book VI
- Mapping/Report Module — Book VII

This is a Blank Page

Table of Contents

ICM MODEL METHODOLOGY PACKAGE ..... a

INTEROFFICE TRANSPORT MODULE 1

Interoffice Transport Flow Chart 1

Network Design/Model Approach 2

Network Components 5

Data Inputs 6

Clustering of End Offices and Tandems 7

Modeling Process 7

Developing Ring Configuration and Calculating Ring Length 7

Calculating Distance from Hosts to Remotes Offices 8

Determining Interoffice Traffic and Sizing Switching Nodes 8

Transport Facilities 9

Outputs 10

APPENDIX A: Data Input Tables 11

InterOffice Transport - Host to Host [XXH2H.DB] 12

InterOffice Transport - Remote to Host [XXH2R.DB] 13

InterOffice Transport - Tandems [XXTANDEMS.DB] 14

InterOffice Transport - InterRing [XXIRING.DB] 15

Materials - Outside Plant Material [XXMATL.DB] 16

Placement Costs - Outside Plant Placement Costs [XXLABR.DB] 17

Processing Flow 18

APPENDIX B: Transport Office Configurations 19

CO Equipment for End Office w/OC-3 Point to Point w/DS-1 20

CO Equipment for End Office w/OC-3 Point to Point w/DS-3 21

CO Equipment for End Office on OC-12 Ring 22

CO Equipment for End Office on OC-48 Ring 23

CO Equipment Tandem or Tandem/Host End Office OC-48 Ring 24

This is a Blank Page

INTEROFFICE TRANSPORT MODULE

The Interoffice Transport Module develops investments for the network components that connect end offices, end offices to remotes or REMX nodes[1], and end offices to tandems. These network components consist of specialized transmission equipment within wire centers and the outside plant facilities that carry communication signals between offices.

The Interoffice Transport investments are passed to the Mapping/Report Module where the associated expenses are calculated to develop monthly costs.

Interoffice Transport Flow Chart

A flow chart depicting how Interoffice Transport investments are determined is shown below. Note that some of the steps in the process take place outside of the model during pre-processing.

[1] REMX refers to CLLI locations where pair gain devices have replaced older technology switches. The Transport Module calculates the investments for the link between the REMX node and its associated host.

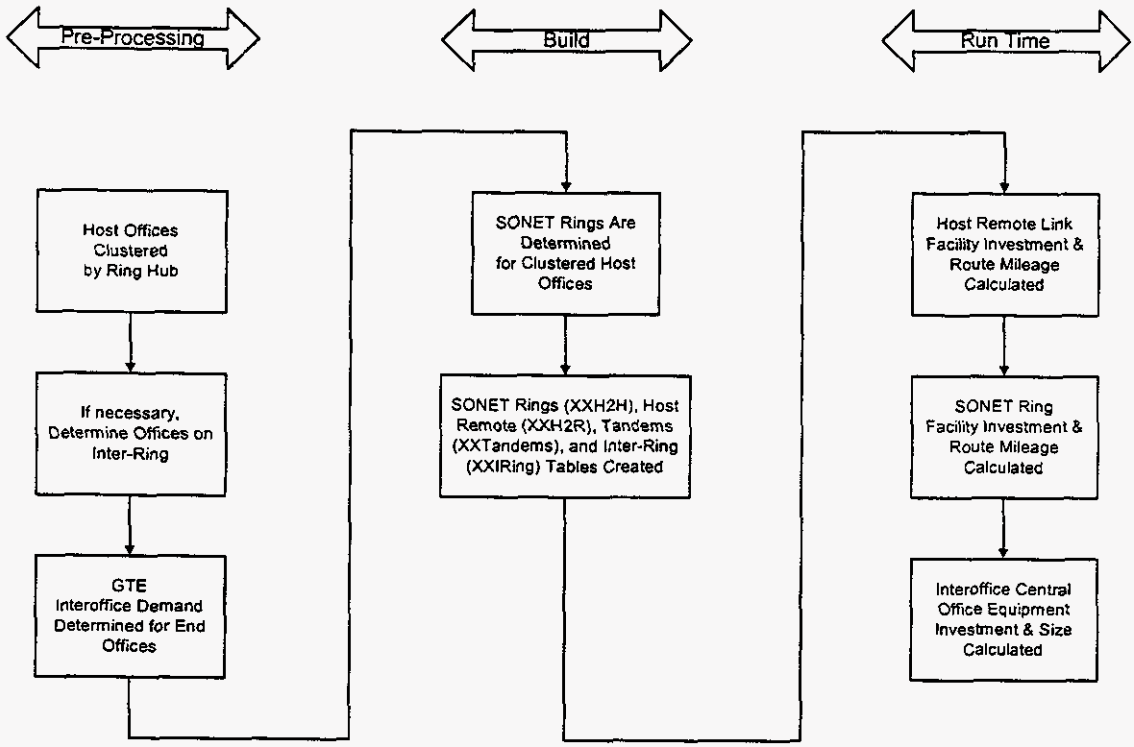


Figure 1: Transport Module Flow Chart

Network Design/Model Approach

The interoffice transport network connects the various switching "nodes" to each other. These nodes consist of end office switches, remote switches and tandem switches. In switching hierarchy, lower level switches are "homed" on switches above them in the hierarchy. Remote switches home on host end office switches, and end office switches home on tandem switches. Previously, switching nodes were connected via "hub and spoke" topology, using direct, point-to-point transmission equipment and outside plant facilities.

Advances in technology have allowed the use of "ring" networks in place of point-to-point connections. SONET (Synchronous Optical Network) rings, using add/drop multiplexers (ADMs) and fiber facilities, are the least cost, efficient technology for interoffice transport. These bi-directional rings interconnect switch nodes and are able to transport signals even if the ring is cut. Figure 2 shows the ring configuration used in ICM when the associated tandem is owned by GTE. It allows connection of up to seven end offices and one tandem switch. If the tandem is outside the franchise area (which would

be the case if GTE did not own the tandem switch), then up to eight end office switches are placed on the ring. The closest office on the ring, known as the gateway, is connected to the tandem using point-to-point facilities as shown in Figure 3.

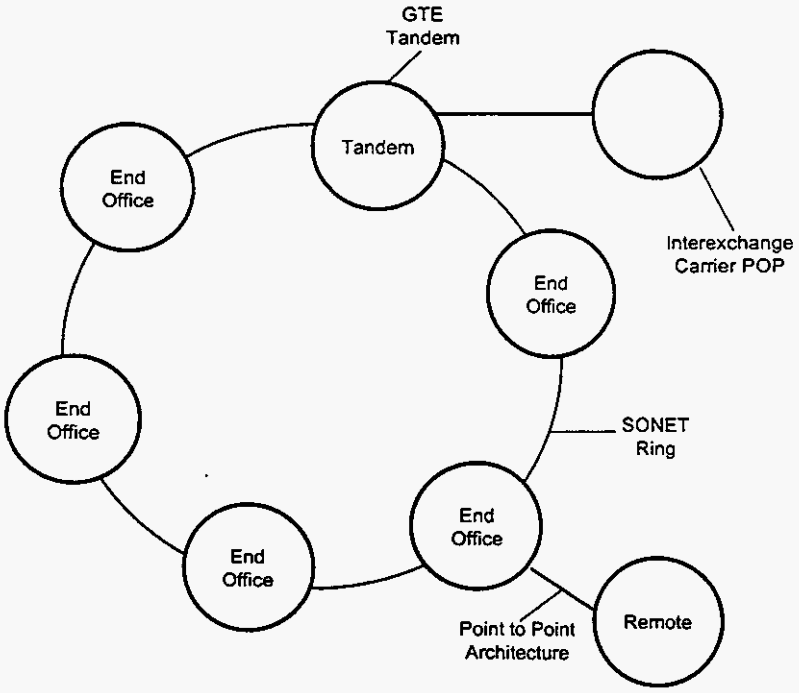


Figure 2: SONET Ring Architecture with GTE Tandem

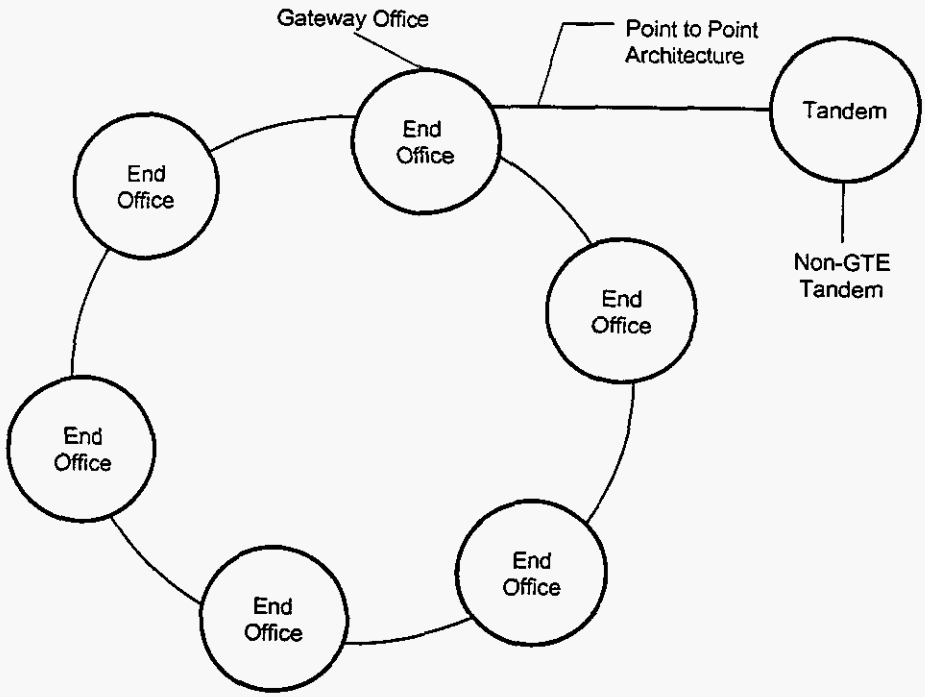


Figure 3: SONET Ring Architecture with Non-GTE Tandem

The Interoffice Transport Module assumes each SONET ring can have a minimum of three and a maximum of eight nodes. If more than eight nodes are connected to a tandem switch, two or more rings are configured. Point-to-point transport facilities are used when only two switching nodes need to be connected. Examples include connections between hosts and remotes, hosts and non-GTE tandems, and two hosts (when only two nodes need to be connected).

The function of each node is to pull traffic from the ring to be terminated at that node, to add on traffic from the node destined for other nodes, and to route "through" traffic which is transiting the node to other nodes on the ring. Because the traffic on the ring enters and exits the node at optical level, it must be converted from optical to electrical signals. These functions are performed by either the add/drop multiplexer (ADM) for ring traffic, or the OC-3 point-to-point system for point-to-point traffic.

Once at the DS-3 or DS-1 level[2], the lines are physically cross connected to their points of termination in the wire center, and in some cases, further demultiplexed to either DS-1 or DS-0 level.

[2] DS-1 multiplexes 24 DS-0 or basic channels, and DS-3 multiplexes 28 DS-1 channels, which is the equivalent of 672 DS-0 channels.

Based on interoffice transport requirements and the available technology, five typical office configurations representing GTE engineering practices have been developed, and these offices are shown in Appendix B. The different offices are composed of some or all of the equipment described in the following Network Components section.

Network Components

The major network components are shown in the five interoffice transport equipment configurations discussed above and shown in Appendix B. The Interoffice Transport Module develops investments for the following network components:

Outside plant facilities include either aerial, buried, or underground fiber cable and support structures such as poles or conduit.

Add Drop Multiplexers (ADMs): Used with OC-12 and OC-48 SONET rings and converts signals between optical and electrical. The electrical signals are at the DS-3 level. ADMs allow traffic from other nodes which terminate at this node to be pulled off the ring. ADMs also allow traffic originating at this node to be added to the ring for distribution to other nodes.

OC-3 Point-to-Point equipment is used for OC-3 level point-to-point transport systems. This equipment converts between optical signals and electrical ones, at either the DS-1 or DS-3 level, depending on demand at the node.

Fiber Distribution Panels serve as the interface between the ADM and the outside plant facilities. Fibers from the outside plant environment are terminated on the panel and connected to the ADM equipment using fiber patch cords.

Channel Banks are multiplexers that combine 24 voice grade and/or data circuits into a DS-1. They are used primarily in offices that require DS-0 special access circuits.

DCSs (Digital Cross-Connect Systems) multiplex and demultiplex electronic signals and act as a means to electronically cross-connect facilities. These are sometimes referred to as Digital Access and Cross-Connect Systems (DACS). DCS 1/0 (between DS-1 and DS-0) and DCS 3/1 (between DS-3 and DS-1) systems are included in ICM in offices that serve as ring hubs.

3/1 Mux used in switch nodes to multiplex and demultiplex between DS-3 and DS-1.

DSX-x (Manual Cross Connect) that allows manual cross connection of facilities. There are two types: DSX-3 for DS-3 level signals and DSX-1 for DS-1 level signals.

Data Inputs

The following items are data inputs to the Transport Module:

Outside plant material and labor - Material costs and placement costs (same as used for the Loop Module) for cable (aerial, underground, and buried), support structures, etc.

Transport equipment - Material costs for Fiber Distribution Panels, Add/Drop Multiplexers, associated DS-3 and DS-1 cards, Point-to-point optical-to-electrical converters, channel banks, cross-connect systems (DCS-x and DSX-x systems), and 3/1 Mux systems.

Switching node data including:

End Office CLLI code

CLLI code for the associated tandem (TCLLI)

CLLI code for the end office that serves as a gateway to an out-of-franchise tandem (GCLLI)

CLLI code for remote offices (RCLLI)

Ring number - designates the clustering of switching nodes determined during pre-processing.

Tandem owner - designates whether the tandem is in-franchise[3] (GTE - G, Contel - C) or out-of-franchise (Other - O).

Number of DS-0, DS-1, DS-3 special access lines associated with each host or remote office.

[3] An in-franchise tandem is one owned by GTE.

Clustering of End Offices and Tandems

During pre-processing, GTE's existing switching hierarchy is used to group offices by tandem areas. End offices (and their tandem if in-franchise) in the same geographic area are usually clustered together to efficiently provision the network.

To determine the office clustering, geographic coordinates are used and a sweep is made from the tandem office in a 360-degree arc. End office switch nodes are identified during this sweep and clustered into groups of seven or less (the tandem is the eighth node for each ring), based on the sweep of the arc. This clusters the nodes for inclusion on rings, with no more than eight nodes on any one ring. A cluster number is input into ICM for each end office to designate the group of nodes with which it is clustered.

Modeling Process

The Interoffice Transport Module:

- develops the SONET ring and point to point configuration
- calculates distance between hosts and remotes (including REMX nodes)
- determines the length of interoffice facilities
- determines the total traffic on each ring and host/remote link and sizes facilities
- determines the equipment configuration at each node
- calculates investments by CLLI code and passes them to the Mapping/Report Module where expense calculations are performed to convert them into monthly costs

Developing Ring Configuration and Calculating Ring Length

ICM examines the end offices and tandems clustered during pre-processing and determines each node's position on the ring. Tandem switch serving areas with more than eight switch nodes require two or more rings, all routed through the tandem switch. In this way, all end office switches are on a ring which includes the tandem. This ensures that traffic between any end office and its host tandem can be carried on a single ring.

If the tandem switch is out of the franchise area, then the non-GTE tandem is not part of a ring, and it is directly connected to the nearest end office, called the gateway office. In this latter case, only end offices are on the ring.

When fewer than three end offices are clustered, the nodes will be linked in a point to point configuration. Remotes and REMX nodes are always directly connected to their end offices in a point to point configuration. In rare cases, a host end office may be directly connected in a point to point configuration to its tandem switch.

After all the interoffice links between nodes are determined for all offices, total length of facilities connecting the nodes is calculated.

The algorithm for interconnecting the nodes on a ring first determines the office closest to the tandem. The link between the tandem and the closest office is the first link in the ring. The next closest office to the tandem is the next node connected to the ring. This process continues until all nodes are included on the ring. The last office must be connected to the tandem to complete the ring.

Calculating Distance from Hosts to Remotes Offices

Based on the homing arrangement identified in the Nodes file the Interoffice Transport Module determines the distance between hosts and remotes. The Nodes file contains the geographic coordinates, longitude and latitude, of the office nodes.

The distance between host and remotes is calculated by combining the fiber feeder routes and interoffice only airline distances that are converted to route distances using the air-to-route mile ratio in ICM.

Determining Interoffice Traffic and Sizing Switching Nodes

Both switched and non-switched traffic are considered when the transport facilities are sized. The demand for DS-1 ports for switched services and the demand for DS-0, DS-1, and DS-3 facilities for non-switched services (special access lines) are included. Thus total interoffice demand is used to size the ring or the point-to-point facilities.

The total demand at each node is included as a GTE specific input for each end office.

The total switched demand is adjusted by an Intra-Ring Factor that reflects the amount of switched traffic that originates and terminates on the same ring. The Intra-Ring Factor is a user option.

The adjusted demand for all switch nodes on a ring determines the minimum size of the ring. Traffic entering and exiting the ring at each node determines

the specific quantity of each equipment item, and the associated equipment investments. This information is passed to the Mapping/Report Module.

Transport Facilities

After the length of all links on the ring and all point-to-point routes are determined, outside plant facilities costs are modeled the same as fiber feeder cable in the Loop Module. The aerial, buried, and underground cable percentages and structure sharing with other utilities that are input for fiber feeder are used to determine interoffice placement investment. Structure investments are also modeled in same as fiber feeder except that interoffice placement investment is adjusted to reflect the facilities shared with fiber feeder routes.

Outputs

The Transport Module outputs are used to develop the monthly costs for transport Basic Network Functions (BNFs). BNFs are then mapped into services or Unbundled Network Elements (UNEs). Typical UNEs for transport are:

- DS-1 to Voice Grade (DS-0) Multiplexing
- DS-3 to DS-1 Multiplexing
- Direct Trunked Transport - Voice Facility
  - Facility per Mile
  - Termination
- Direct Trunked Transport - DS-1
  - Facility per Mile
  - Termination
- Direct Trunked Transport - DS-3
  - Facility per Mile
  - Termination
- Common Transport
  - Transport - Termination - Set Up
  - Transport - Termination - MOU
  - Transport - Termination - Average MOU
  - Transport - Mile - Setup
  - Transport - Mile - MOU
  - Transport - Mile - Average MOU

APPENDIX A: Data Input Tables

The data input tables are listed below. The tables can be viewed and printed under the menu options **Tables/Interoffice**. The name of each database file is listed; often the two letters indicate state (that is in, xxexbase.db, 'xx' is a state's abbreviation). A user may modify data in all of the tables listed below. Detailed descriptions of each of the tables are provided in the following section.

Following the description of data input tables, a "bubble chart" for Interoffice is provided under the heading 'Processing Flow'. The chart contains a graphic presentation of the database files that are inputs to and outputs of the Interoffice Transport Module.

Interoffice

- Interoffice
  - Host to Host (XXH2H.DB)
  - Remote to Host (XXH2R.DB)
  - Tandems (XXTANDEMS.DB)
  - InterRing (XXIRING.DB)
- Materials
  - Outside Plant Material (XXMATL.DB)
- Placement Costs
  - Outside Plant Placement Costs (XXLABR.DB)
- Processing Flow
  - Interoffice

InterOffice Transport - Host to Host [XXH2H.DB]

Purpose: Provide the data necessary to engineer the interoffice ring and point to point structure and determine the required investment

COLUMN HEADING	DESCRIPTION
ECLLI	End office CLLI code
HUBCLLI	CLLI code for the hub of the ring(s)
GCLLI	CLLI code for the end office that serves as a gateway to an out-of-franchise tandem
RING	Number designating on which ring the end office is located. Determined during pre-processing.
REMOTESW	Number of remotes associated with the end office that serves as a host. Calculated by ICM.
SWPORTS	Number of DS-1 switch ports associated with end office.
DS0SAL	Number of DS0 - number of special access lines associated with the end office
DS1SAL	Number of DS1 - number of special access lines associated with the end office
DS3SAL	Number of DS3 - number of special access lines associated with the end office

InterOffice Transport - Remote to Host [XXH2R.DB]

Purpose: Provide the data necessary to determine the interoffice transport investment required for remotes to host offices.

COLUMN HEADING	DESCRIPTION
HCLLI	Host switch CLLI code
RCLLI	Remote switch CLLI code[4]
SWPORTS	Number of DS-1 switch ports associated with the remote office.
DS0SAL	Number of DS0s - number of special access lines associated with the remote office
DS1SAL	Number of DS1s - number of special access lines associated with the remote office
DS3SAL	Number of DS3s - number of special access lines associated with the remote office

[4] If remote CLLI has been replaced by a pair gain device or REMX (see xxnodes.db, Substitution column), DLC investments are pulled from the Outside Plant Material table and no central office equipment is placed; only the investment required for the transport facilities is calculated.

InterOffice Transport - Tandems [XXTANDEMS.DB]

Purpose: Provide the data necessary to determine the interoffice ring and point to point structure and determine the required investment.

COLUMN HEADING	DESCRIPTION
TCLLI	CLLI code for the associated tandem
COLOCLLI	CLLI code for the end office that is colocated with the tandem
TNDCOMPANY	Tandem owner – designates whether the tandem is in-franchise (GTE-G or Contel-C) or out-of-franchise (Other-O)
LAT	Latitude of tandem
LONG	Longitude

InterOffice Transport - InterRing [XXIRING.DB]

Purpose: Provide the data necessary to engineer the interring and determine the required investment.

COLUMN HEADING	DESCRIPTION
ICLLI	CLLI code for the office associated with the interring
ICLLI SUBSTITUTION	CLLI code for the office that is colocated with the ICLLI office
ICLLI HUB	CLLI code for the office that is colocated with the interring hub office
ICLLI HUB SUBSTITUTION	CLLI code for the office that is colocated with the interring hub office
IRING	Number designating which interring the office is located. Determined during pre-processing
ISWPORTS	Number of DS-1 switch ports associated with interring office

Materials - Outside Plant Material [XXMATL.DB]

Purpose: Lists the unit prices for outside plant materials. This table is used in both the Loop and Transport Modules. See the outside plant material table included in the Loop Outside Plant Material [XXMATL.DB] input description in the Loop Module Appendix A Book II within the ICM Methodology package[5].

COLUMN HEADING	DESCRIPTION
MATERIAL TYPE	Abbreviation for the type of equipment (for example, CUBUR for copper buried cable and POLE30 for a 30-foot pole). If the material is made up of multiple components, they are listed accordingly in the OSP Material table referenced previously in this Appendix.
SIZE	Size of the outside plant material (for example, 600 for a 600-pair underground copper cable)
DESCRIPTION	Brief description of the equipment. The purpose of each type of equipment is defined in the OSP Material table referenced previously in this Appendix.
UNIT PRICE	Investment in the equipment per unit. This cost includes all components associated with any defined material type.
MINOR MATERIAL	Indicates Major or Minor Material Item (True or False). If True, the item is classified as Minor and its cost is not used towards network investment. If False, the item is classified as Major and its cost is used towards network investment.

[5] ICM Model Methodology package contains seven booklets: Conceptual Framework (Book I); Loop Module (Book II); Switch Module (Book III); Interoffice Transport Module (Book IV); SS7 Module (Book V); Expense Module (Book VI); and Mapping/Report Module (Book VII).

Placement Costs - Outside Plant Placement Costs [XXLABR.DB]

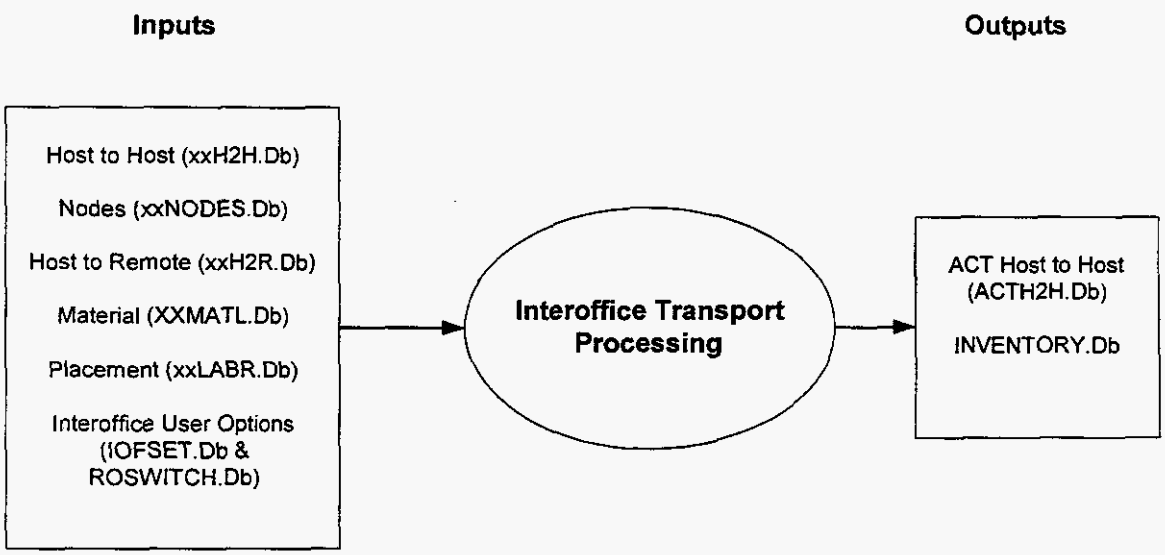
Purpose: Provides labor costs for installing specific types of equipment. The labor requirements and tasks included in each cost are defined in the outside plant placement table listed in the Loop Module Appendix A Book II within the ICM Methodology package. In states where there are significant cost differences between areas, two sets of weighted average rates are developed (Tier A and Tier B).

COLUMN HEADING	DESCRIPTION
LABOR TYPE	Type of labor required, as identified by an alphanumeric code (for example, LP01A for placing a pole). The tasks involved in each type of labor are defined in the OSP Placement table referenced previously in this Appendix.
DESCRIPTION	A description of the work done (for example, "place anchor & guy"). These descriptions are extracted from the contracts used by GTE and the contractor. They specifically identify the tasks that are included in any labor requirement.
TIER A RATE	Labor activity rate for Tier A areas. Used by ICM when FALSE appears in the Tier B column in the XXNODES.DB table.
TIER B RATE	Labor activity rate for Tier B areas. Used by ICM when TRUE appears in the Tier B column of the XXNODES.DB table.

Processing Flow

The Interoffice bubble chart provides a graphic view of the database files that are inputs to and outputs of the Interoffice Transport Module.

Interoffice Transport Bubble Chart

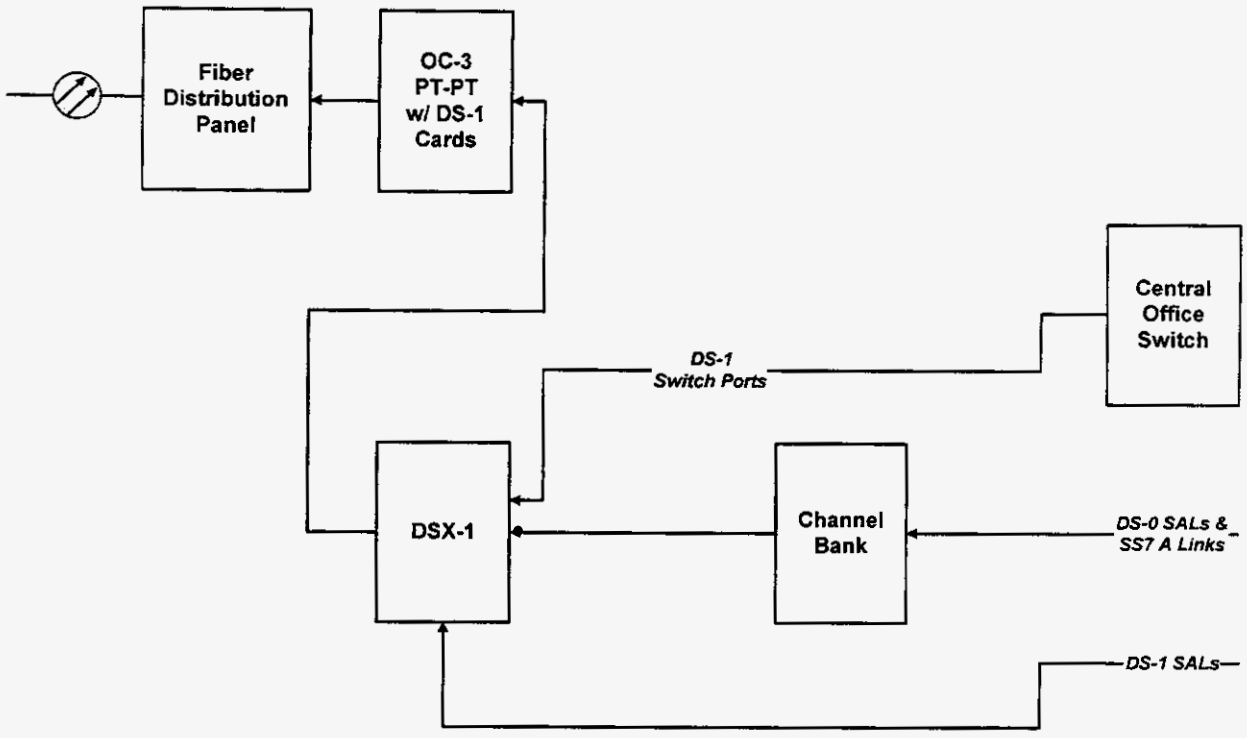


APPENDIX B: Transport Office Configurations

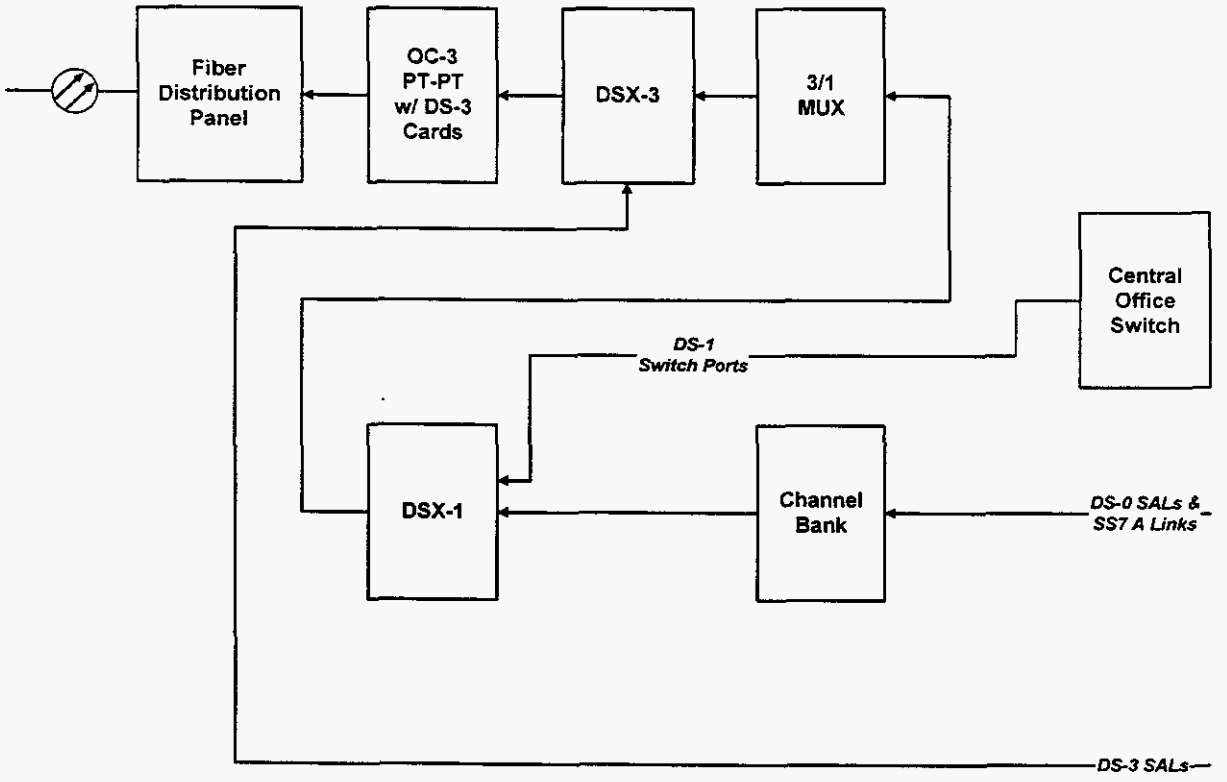
Transport Office Configurations:

- CO Equipment for End Office w/OC-3 Point to Point w/DS-1
- CO Equipment for End Office w/OC-3 Point to Point w/DS-3
- CO Equipment for End Office on OC-12 Ring
- CO Equipment for End Office on OC-48 Ring
- CO Equipment Tandem or Tandem/Host End Office OC-48 Ring

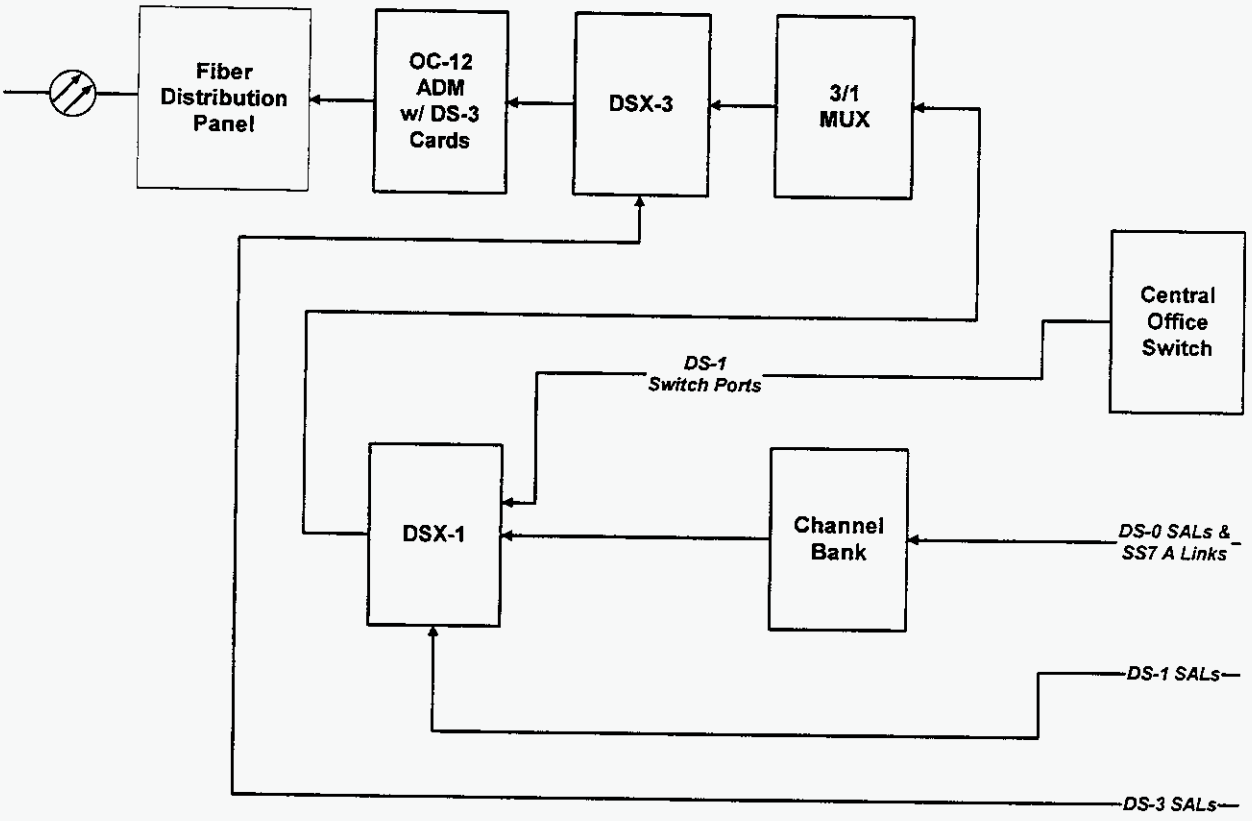
CO Equipment for End Office w/OC-3 Point to Point w/DS-1



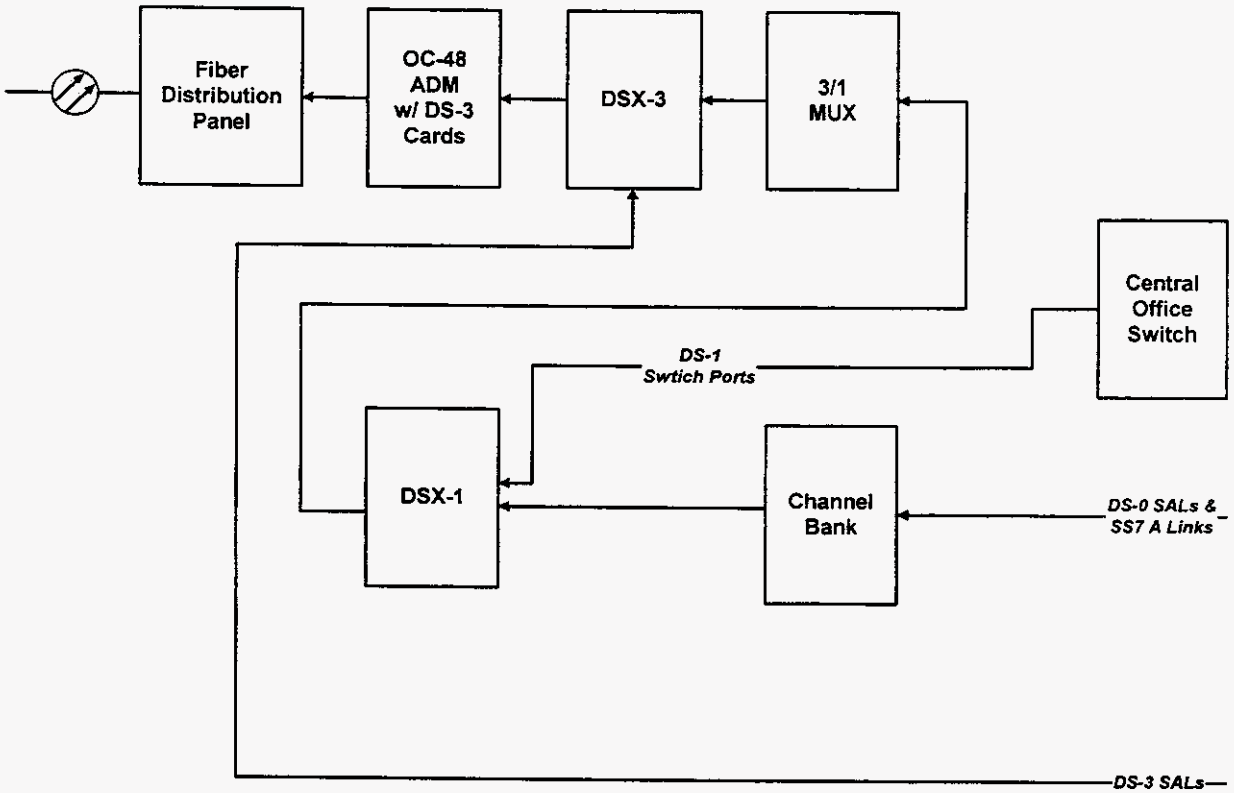
CO Equipment for End Office w/OC-3 Point to Point w/DS-3



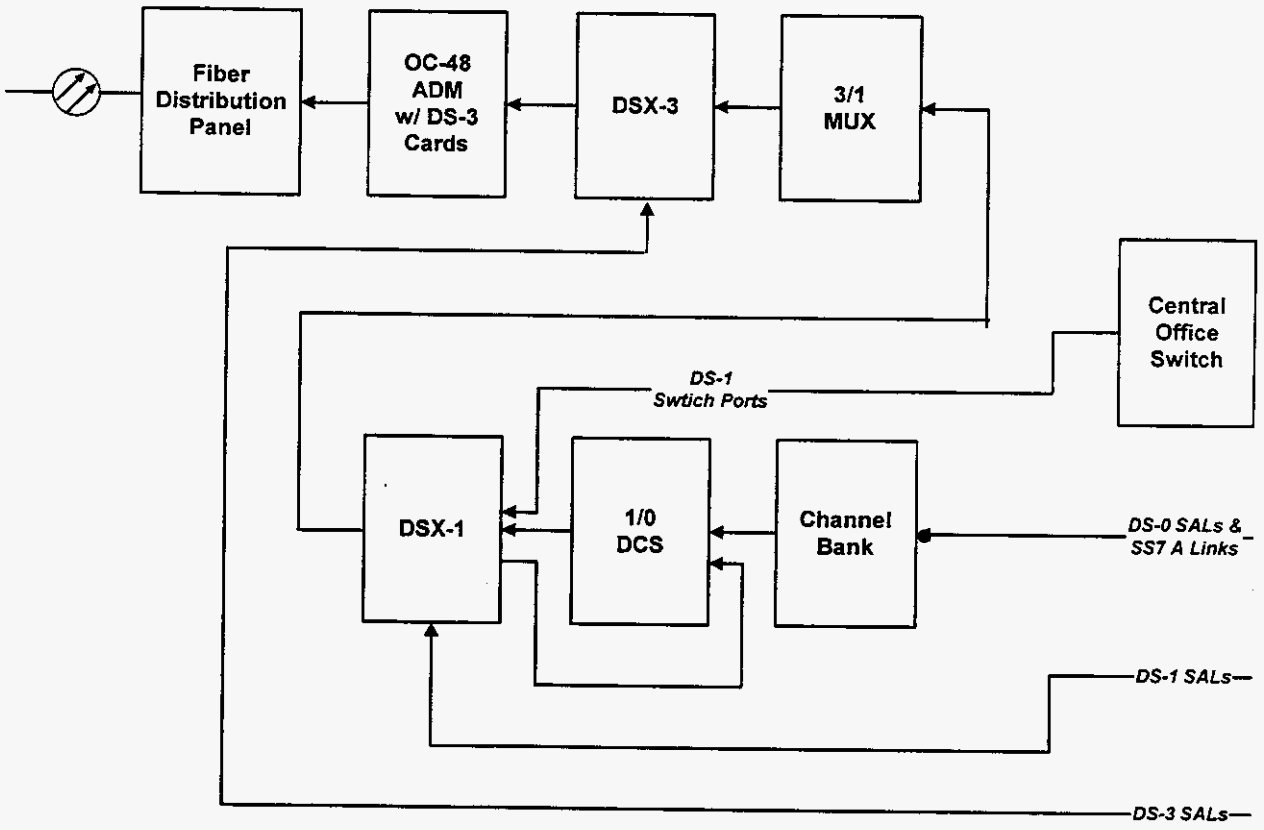
CO Equipment for End Office on OC-12 Ring



CO Equipment for End Office on OC-48 Ring



CO Equipment Tandem or Tandem/Host End Office OC-48 Ring



This is a Blank Page

This is a Blank Page



Integrated
Cost Model

Release 4.1

Model Methodology

SS7 Module

(Book V of VII)

GTE Service Corporation

Copyright © GTE 1997 - 2000. All rights reserved.

This is a Blank Page

ICM MODEL METHODOLOGY PACKAGE

- Conceptual Framework — Book I
- Loop Module — Book II
- Switch Module — Book III
- Interoffice Transport Module — Book IV
- SS7 Module — Book V
- Expense Module — Book VI
- Mapping/Report Module — Book VII

This is a Blank Page

Table of Contents

ICM MODEL METHODOLOGY PACKAGE ..... a

SS7 MODULE 1

Network Components 1

Modeling Approach 4

SSP Modeling 5

STP Modeling 5

SCP Modeling 7

Link Modeling 7

A Links between SSP and STP 8

D Links between Local and Regional STPs 9

D Links between Regional STPs 9

A Links between Regional STPs and SCPs 10

Data Inputs 10

Pre-processing 10

Model Processing 11

Outputs 11

APPENDIX A: Data Tables 14

Data Input Tables (SS7) 15

Service Switching Point (SSP) [XXSSP.DB] 15

Signal Transfer Points (STP) [XXSTP.DB] 16

STP Location [XXSTPLOC.DB] 17

SS7 Investment [XXSS7INV.DB] 18

SS7 Investment Table Item Descriptions 19

Loading Factor [XXLOAD.DB] 21

Network Parameters [XXSS7PAR.DB] 21

SS7 Network Parameters Table Item Descriptions 22

Data Output Tables (SS7).....24
SS7 Host Component Output [ACTSS7H] 24
SS7 Module Output Descriptions (In ACTSS7H and ACTSS7R Tables) 25
Processing Flow.....34

SS7 MODULE

The Signaling System 7 (SS7) network is a stand-alone network used for the transmission and distribution of signals that tell the switched telephone network how to operate. It is connected to the switched telephone network at each end office and tandem and acts primarily to set up and disconnect calls and to control the network for efficient utilization of facilities. The SS7 network is also used to query databases such as for 800/888 and credit card calls.

The SS7 Module[1] identifies the SS7 investments required for each end office. These investments are passed to the Mapping/Report Module where they are combined to determine the investment required for the use of SS7 for call set up and unbundled network elements. The associated expenses are then calculated and the monthly costs developed.

Network Components

Figure 1 shows the basic components of the SS7 network.

[1] The SS 7 Module Methodology is (Book V of VII) within the ICM Methodology package. The contents of the package include seven booklets: Conceptual Framework (Book I); Loop Module (Book II); Switch Module (Book III); Interoffice Transport Module (Book IV); SS7 Module (Book V); Expense Module (Book VI); and Mapping/Reports Module (Book VII).

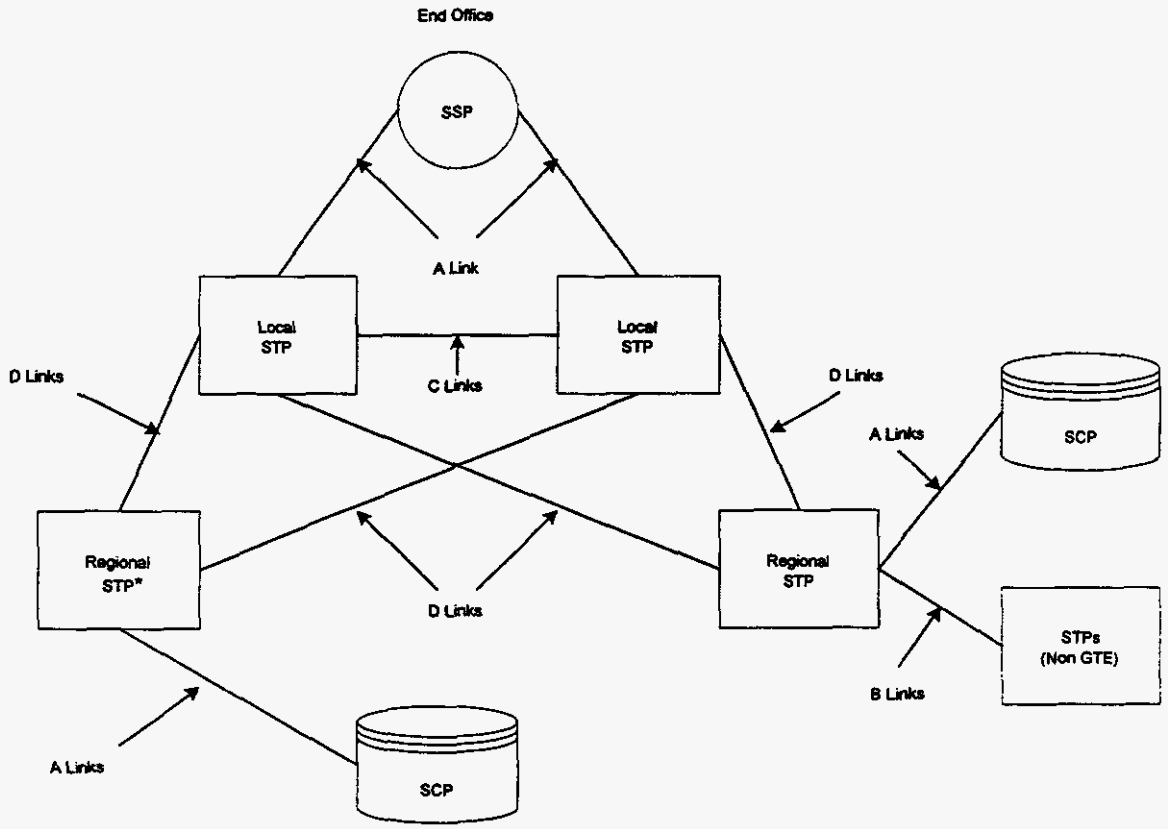


Figure 1: Signaling System 7 Network Components

In ICM, investment for the following types of SS7 equipment are identified:

- **Service Switching Points (SSPs)** are the SS7 network elements contained within end offices and tandem offices. SSPs provide the points of connection between the switches and the SS7 network links.
- **Service Control Points (SCPs)** are remote databases that provide translation and routing information for advanced network services. SCPs are collocated with regional Signal Transfer Points (STPs).
- **Signal Transfer Points (STPs)** are the SS7 components that send call setup and disconnect information between offices and route data queries between offices and SCPs.
- **"A," "C," and "D" Links** are DS-0 level transport circuits used to carry data between the different points (SSPs, SCPs, and STPs) in

* Regional STP and SCP are collocated.

the SS7 network. Depending on SS7 network traffic volume, links can be common or specific to SS7-related services.

- Each SSP has at least one pair of A links, one to each local STP of a mated pair.
- Each pair of local STPs has a set of four D links to a pair of regional STPs.
- A regional STP is connected by at least eight A links to each SCP of a mated pair.
- Each mated pair of STPs is connected by between two and eight C Links.

- **STP Ports** for connection of "A," "C," and "D" Links. These ports can be basic, common, or specific for each SS7-related service.

The SS7 Module assumes that each end office switch is equipped as an SSP having at least one A link to each local STP of a mated pair. STPs are linked to regional STPs that are connected to SCPs and to STPs for other networks. Software is used to control this network and the associated databases.

For basic calls, end office switches send setup messages about the dialed telephone number from the SSP to the local STP. The STP determines how the call is to be routed and reserves a path through the switched network for the call. This information is in turn relayed to each switch needed for the call, which may include a tandem switch. The SS7 network ensures efficient use of the switched network by not setting up call paths until they can be used for actual calls.

The SS7 network also is used for other services requiring database queries, including 800/888 Service number translations and LIDB (Line Information Data Base) queries.

The complete list of SS7 network elements can be found under the heading "Outputs" on page 11.

Modeling Approach

The SS7 Module is constructed as a forward-looking model of GTE's national Signaling System #7 network. The model equations are designed for application to any state, while user inputs include the appropriate data for producing accurate state-specific results. See the following flowchart for an overview of model development and processing steps.

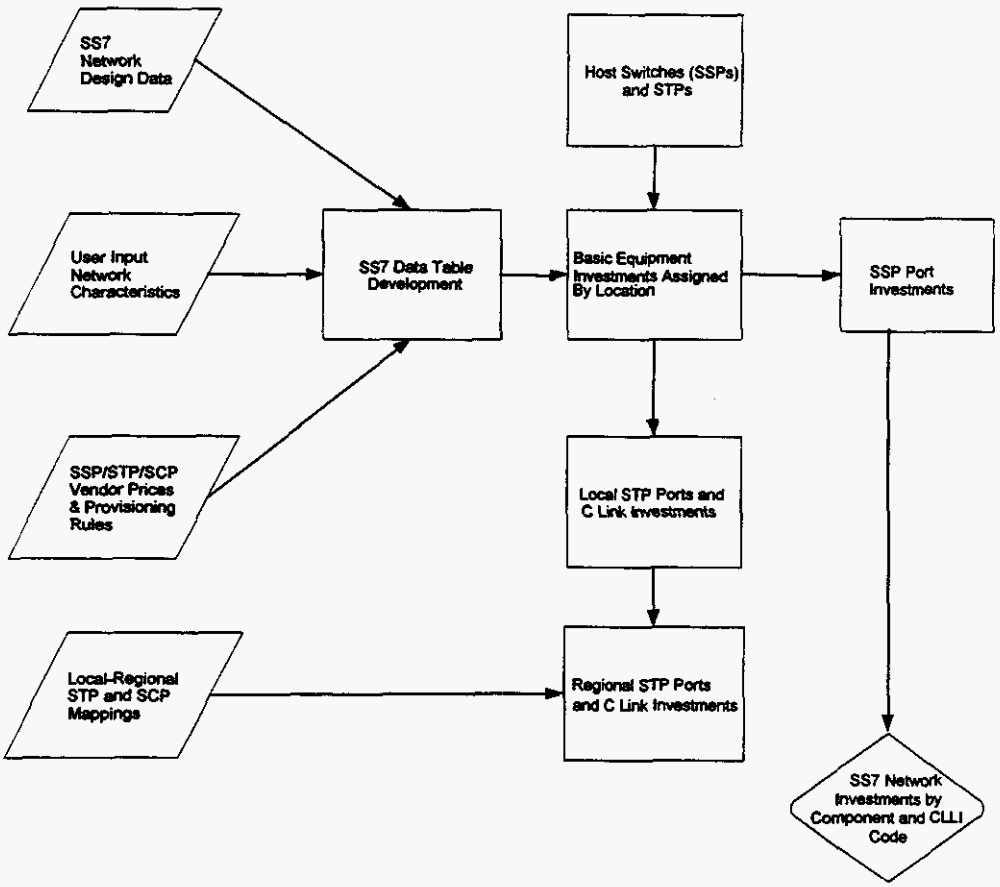


Figure 2: SS7 Flow Chart

Model development begins with a study of the SS7 network, including the engineering of SSPs, STPs, SCPs, and all of the links that connect these nodes. Engineering data is gathered from switch, STP, and SCP vendors and from GTE network planners. The Telcordia switching model SCIS and GTE's COSTMOD produce SSP hardware investments, while GTE-vendor contracts include software investments. Contracts and engineering documents

enable GTE to model the various versions of STPs and SCPs. The ICM Transport Module provides circuit equipment and cable investments for links, such as C and D links that are shared among offices. Since A link investments are office-specific, those investments are calculated within the Mapping Module. Details for each part of the model development are given below.

SSP Modeling

Each host and standalone switching office is assumed to be equipped as a Service Switching Point (SSP). Thus, the model includes ports for at least two A links between an office and the pair of STPs serving that office. Based on traffic requirements, some offices may have ports for more than two A links. The actual number of A links, from GTE Trunk Traffic Engineering (TTE) data, is stored in the SSP Table for each office.

The SS7 Investment Table includes switching software investments for SSP ports and for processing 800 database queries. The variable "SSPIPort" is the software investment per port for the initial two ports. The variable "SSPAPort," investment per additional port, is also used if the office has more than two A links. Since no software is provisioned on a per link basis, all basic SSP software is included currently in the investment for the two initial links. These port investments are included in the costs for call setup and in the service costs for database queries. The software required for 800 database queries is identified by the Investment Table variable SSP800_SW; this software is included only in the service-level costs for 800 database queries.

The SCIS and COSTMOD switch investment models include SSP hardware as a basic switch investment. Since outputs from these models are used in the switch module, the SS7 module does not include them.

SSP software investments are stored in the SS7 Investment Table. Engineering and installation investments are added using the Engineered, Furnished, and Installed (EF&I) factors from the Switching Module.

STP Modeling

Signaling Transfer Points are modeled according to functional cost drivers, resulting in the following investment categories and corresponding SS7 Investment Table items:

Investment Category	Investment Table Variable
Direct Investment per STP Port	STPPort
STP Common Equipment	STPCommon
STP Global Title Translation Software	STP_GTTSW

The Direct Investment per STP Port and the STP Common Equipment Investment make up the switching portion of the investment per STP port (circuit equipment and outside plant portions are addressed later). The

common equipment investment is converted to common investment per port through division by number of ports. The number of ports is the STP port capacity for volume-sensitive costs and the number of ports in service for average costs; these quantities are stored in the STP Table for each STP location.

Global Title Translation (GTT) software is required to process a Transaction Capabilities Applications Part (TCAP) message for 800, LIDB, or LNP database queries. The total software investment is stored in the SS7 Investment Table, and the model calculates an investment per message by dividing the investment by number of TCAP messages. For volume-sensitive costs the number of messages is the message capacity, from the SS7 Network Parameters Table, times the number of ports in service. For average costs this quantity is adjusted by the average utilization on the links. At local STPs the A link utilization is used, and at regional STPs the D link utilization is used.

The STPs, which are modeled, are DMS STP, the DSC A32 and the DSC Mini STP. The DMS STP can support up to 432 links, the DSC A32 STP can support up to 720 links and the DSC Mini STP can support up to 120 links.

STP prices and provisioning rules are used to develop a "bottoms-up" model for each type of STP. That is, the specific categories of investments (port, common, GTT) consist of equipment items and quantities according to the vendor's provisioning rules for each item. Interface cards and related shelves and frames are typically included in the direct investment per port. Other items are provisioned simply as a specific quantity per STP, such as central processors, maintenance equipment, and spare circuit packs. This equipment is generally considered common. In addition, all software related to database queries is included in the GTT Software investment.

The vendors, DMS and DSC, have specified a price per line that includes all the required STP equipment. An effective discount is developed based on this price per line and the equipment item list prices, resulting in discounted STP investments by category.

For STP values in the SS7 Investment Table, discounted STP investments are combined with investments for miscellaneous hardware, such as test equipment, routers, and workstations.

Hardware and software STP investments are stored in the SS7 Investment Table, along with specific quantities of hours for engineering, installation, and installation test labor. These hours are used with the labor rates stored in the SS7 Loading Factor Table in order to add engineering and installation investments to the hardware and software investments. Since an STP may or may not be in the same state as the end offices, each STP's state abbreviation is stored in the STP Table so that the model can use the state-specific labor rates stored in the SS7 Loading Factor Table.

SCP Modeling

For Service Control Points the model uses investments for IBM hardware and Telcordia software. Hardware and software SCP investments are stored in the SS7 Investment Table, along with specific quantities of hours for engineering, installation, and installation test labor. ICM uses these hours with the state-specific labor rates stored in the SS7 Loading Factor Table in order to add engineering and installation investments to the hardware and software investments.

For each type of database query (DB800, LIDB, and LNP), the model computes an SCP investment per query. Investments are based on query capacities for volume-sensitive costs and on estimated query demand for average costs. Volume-insensitive investments are defined as the difference between average and volume-sensitive investments.

SCP query capacities in the GTE SS7 network are defined as follows.

- Average number of A links between a regional STP and an SCP is 16.
- Each A link has a busy hour engineered capacity of 10, 080,000 octets.
- Total SCP octet capacity is 16 * 10,080,000 (or 161,280,000) octets.
- Value of 161,280,000 octets is divided by 94 octets[2] per query.
- As a result, the capacity for DB800 queries is estimated at 1,715,744 per SCP per busy hour.

The database query demand per SCP is estimated by multiplying the above capacity by the SCP utilization factor estimate from the SS7 Network Parameters Table.

Link Modeling

ICM models the following links in the SS7 network:

- A Links between SSP and Local STP
- D Links between Local STP and Regional STP
- D Links between Regional STPs
- A Links between Regional STPs and SCPs

The model also considers investments for C links between the mated pairs of STPs, but those links have a fixed quantity for each STP pair, which is not specific to any particular service or network element. In fact, the C links are

[2] As per the SS7 Network Parameter Table, the number of octets per query is 94.

used only for network management. Therefore, the model treats these links as shared investments for each pair of STPs.

Circuit equipment investments per link at each end of the C links are computed consistently with the mapping module calculation for a dedicated DS0 termination. For C links within a GTE LATA, DS0 facility mileage investments are also calculated and multiplied by link mileages. For InterLATA C links the model uses the monthly lease expense.

For STP pairs within the state being studied, circuit equipment and fiber investments are taken from the transport module outputs for the respective wire center. The model identifies the wire center associated with the STP through the STP Table field "WCenter_CLLI", which is the CLLI code of the wire center switch colocated with the STP. For out-of-state STPs, such as regional STPs, the model uses the average of the transport outputs for the state being studied. This method provides for consistency of C link investments among all offices studied in a state.

Detailed descriptions for modeling of all A and D links follow.

A Links between SSP and STP

The A links between SSPs, or end offices, and STPs, is modeled both for the unbundled network element "SS7 Link" and for the SS7 call setup elements.

The unbundled SS7 link is modeled according to its wholesale rate elements, which consist of the Link Termination and the Link Facility Mileage. The Link Termination cost includes the investments of all circuit equipment required for a DS0 termination at both the SSP and the STP.

Office-level facility costs per mile for unbundled SS7 links are calculated within the Mapping Module using the same equations as for dedicated DS0 transport facilities. At the state level, the office-specific results are averaged using number of links times the total mileage of the two links as the weighting factor. Link facility costs apply only to those offices that are linked via GTE facilities to the pair of local STPs. Thus, link facility costs are not calculated for offices using leased links to connect to the STPs. Generally, InterLATA links are leased, while most IntraLATA links are on GTE facilities.

For SS7 call setup network elements, the link termination components are combined with the link facility components within the Mapping Module. The link facility component investment is multiplied by the route mileage between the SSP and the STP so that both termination and facility investments are identified on a per link basis. The combined investment per link is then divided by octets per link and multiplied by octets per call to obtain the link investment per call setup.

D Links between Local and Regional STPs

The D links that connect local STPs to regional STPs are used for transporting database query messages to SCPs. The SS7 Module calculates investments for those links and includes them in the basic network function cost for local-regional SS7 transport. The D link costs are also included in the service costs for database queries.

The D link costs consist of the costs per DS0 for interoffice circuit equipment at both ends, and the interoffice facilities between the STPs.

If the local and regional STPs are in the same state, then the STP-specific circuit equipment investments are used, based on the transport module outputs for the wire centers where the STPs are located. If the local and regional STPs are in different states, then the model uses the statewide-average circuit equipment investments and units from the transport module outputs. This method produces a consistent set of D Link investments among all offices in a state.

With few exceptions, the local and regional STP pairs are in different LATAs. In those cases the interoffice facilities are modeled as leased links. The monthly D link lease costs are stored in the STP Table by STP. When the local and regional STP pairs are in the same LATA, the links are modeled as GTE facilities using an average of the DS0 transport facility costs at the two STPs. The transport facility costs at the STPs are based on the transport module outputs for the wire centers where the two pairs of STPs are located.

D Links between Regional STPs

The D links connecting two pairs of regional STPs transport database query messages across regions. These links are used when the SCP and the state being studied are in different regions. The SS7 Module calculates investments for these links and includes them in the basic network function cost for region to region SS7 transport. These D link costs are also included in the service costs for database queries.

The D link costs consist of the costs per DS0 for interoffice circuit equipment at both ends and the interoffice facilities between the STPs.

Since no two pairs of Regional STPs are in the same state, the model uses the statewide-average circuit equipment investments and units from the transport module outputs.

All combinations of regional STP pairs are also in different LATAs. Therefore, the region to region interoffice facilities are modeled as leased links. The monthly D link lease costs are stored in the STP Table by STP and by service (DB800, LIDB, and LNP).

A Links between Regional STPs and SCPs

The A links between regional STPs and SCPs are included in both network element and service-related query costs. Each SCP is collocated with a regional STP. On average, 16 A links are used between a regional STP and an SCP. In the case of the SCP collocated with an STP, the A links cost is zero because no interoffice facilities are required. In this case the cost is for Channel Bank terminations only. For the SCP collocated with the other STP in the mated pair, the A link is equivalent to the C link between the STPs, with the same circuit equipment and interoffice facility costs. Therefore, the A link cost used is half of the total cost of C links between the mated pair of regional STPs. If the regional STPs are in the same state as the state being studied, then STP-specific transport costs are used for these A links. Otherwise, the model uses statewide average component outputs from the transport module, along with the specific distances between the STPs.

Data Inputs

SS7 data inputs consist of six tables that contain all investment, expense, usage, engineering, and geographic data necessary to model the SS7 network. The Investment Table includes all SSP, STP, and SCP equipment and software utilized. Lease expenses are entered in the SSP and STP tables for all SS7 links not owned by GTE. Usage and engineering parameters, such as octets per call and octets per link, are in the Network Parameters Table. Geographic information, such as STPs serving SSPs and distances between STPs, is stored in the SSP and STP tables. The model also includes a loading factor table containing state-specific hardware and software minor material loadings and labor rates for all states in the SS7 network of the state being studied.

Appendix A contains detailed information about the data inputs and their impacts on the outputs.

Pre-processing

Based on network requirements and engineering planning guidelines, the number and type of each component used in the SS7 network is determined external to the model. The state-specific forward-looking local telephone network and the forward-looking national SS7 network are the bases for identifying the required investments and characteristics. Each end office is assumed to be an SS7-equipped digital switch having A links directly to a mated pair of STPs to ensure reliable service. Hardware and software investments for each SS7 component are developed through detailed modeling of the engineering for each switch technology type (DMS-100, 5ESS, etc.) and each STP type (Nortel DMS STP or a DSC version). These investments are entered according to the appropriate unit cost driver, e.g. port, link, query, etc.

All applicable link lease expenses are calculated during pre-processing and entered into the SSP (for SSP-STP A links) and STP (for C and D links) tables.

Model Processing

The SS7 Module calculates the component investments by CLLI code required by the Mapping Module for cost studies of SS7-related network elements and services. Following is a summary of the steps involved after all of the pre-processing and entry of data inputs.

- Investment Table values are combined with the minor material factors, Switch EF&I factors, and labor rates from the Loading Factors Table to determine total investment for each table item.
- STP link and port investments and units are calculated for each local STP in the state's network.
- STP link and port investments and units are calculated for each regional STP serving the state.
- SSP-specific link and port investments and units are calculated for each host/standalone office in the SSP Table.
- Local and regional STP investments and units are assigned to the SSPs according to the SSP-STP serving arrangements listed in the SSP Table.
- The ACTSS7H host component output file is populated with the above results by host/standalone CLLI code.
- The ACTSS7R remote component output file is populated with the respective host results for each remote.
- The ICM system-wide component output table, ACTCMPNT, is populated with the results from ACTSS7H and ACTSS7R.

Outputs

The SS7 Module produces component investments by CLLI code for the following types of costs:

Average Cost- based on total costs and the estimated actual average busy hour usage of the network elements. Usage is estimated at 25 percent of usable capacity[3]

Volume Sensitive Cost- based on the maximum usable capacity of the network element, and including only equipment having investments that vary with demand.

[3] Usable capacity is defined as 10,080,000 octets per link per busy hour. All other usage-sensitive capacities are derived from this value, which is based on 40 percent engineered maximum occupancy on 56 Kbps links.

Volume Insensitive Cost- based on the part of the average cost that is not recovered by the volume sensitive cost at average usage. It is the difference between total cost and volume sensitive cost.

Appendix A contains details about the SS7 Module component outputs and their impact on network element and service cost outputs.

The Mapping/Reports Module uses component investments for the categories listed above to produce total investments and unit costs for the following Basic Network Functions (BNF)s:

- Investment per SS7 link
  - Termination
  - Facility Mileage
- Investment per STP Port
- Investment per data base query
  - 800/888 Service
  - LIDB
  - LNP
- SS7 Dedicated Transport
  - Local STP - Regional STP
  - Regional STP - Regional STP
- SS7 Call Setup
  - By rate period

The SS7 Cost per Call Setup listed above is in addition to the call setup cost calculated within the Switching Module. The Switching Module provides the cost components for call setup within the switch, including the SSP hardware, while the SS7 Module provides the components for SSP software, SSP-STP links, and STP ports. The Mapping Module uses ICM usage parameter inputs to convert SS7 Module outputs from busy hour investments to costs for any call in a any day.

The Mapping Module uses the above BNF outputs to calculate the following SS7-related Unbundled Network Element (UNE) costs:

- Investment per SS7 link
  - Termination
  - Facility Mileage

- Investment per STP Port
- Investment per data base query
  - 800/888 Service
  - LIDB
  - LNP
- SS7 Dedicated Transport
  - Local STP - Regional STP
  - Regional STP - Regional STP
- Switching Call Setup
  - By rate period

The Switching Cost per Call Setup listed above combines the switch and SS7 call setup costs for a total investment per call setup

The Mapping Module also uses the SS7 BNF outputs to calculate product/service costs for call setup and for database queries.

APPENDIX A: Data Tables

The data **input** and **output** tables are listed below. The name of each database (db) file is listed below; often the first two letters indicate state (that is, xxexbase.db). A user may modify data in all of the tables listed below. Detailed descriptions of each of the tables are provided in the following section.

Following the description of data input and output tables, a bubble chart for SS7 is provided under the heading 'Processing Flow'. The chart contains a graphic presentation of the database files that are input to and outputs of the SS7 Module.

SS7 Data Input Tables

- Service Switching Point (SSP) [XXSSP.DB]
- Signal Transfer Points (STP) [XXSTP.DB]
- STP Location [XXSTPLOC.DB]
- SS7 Investment [XXSS7INV.DB]
  - SS7 Investment Table Item Descriptions
- Loading Factor [XXLOAD.DB]
- Network Parameters [XXSS7PAR.DB]
  - SS7 Network Parameters Table Item Descriptions

SS7 Data Output Tables

- SS7 Host Component Output [ACTSS7H.DB]
- SS7 Remote Component Output [ACTSS7R.DB]

Processing Flow

- SS7 bubble chart

Data Input Tables (SS7)

The data input tables are listed below. The input tables can be viewed and printed under the menu option **Tables/SS7.**

Service Switching Point (SSP) [XXSSP.DB]

Purpose: Provides CLLI Code for Service Switching Points (SSP) in the state, along with the CLLI codes of STPs supporting the SSPs. As part of a forward-looking model, each host/remote office is assumed to be SSP equipped.

COLUMN HEADINGS	DESCRIPTION
SSP_CLLI	The CLLI code for each host end office in the state
LOCAL_STP1	Local STP 1 CLLI code
LOCAL_STP2	Local STP 2 CLLI code
ALINKS	Number of A Link Ports (minimum of two ports)
A1 EXPENSE	Monthly Lease Expense to Local STP 1
A2 EXPENSE	Monthly Lease Expense to Local STP 2

Signal Transfer Points (STP) [XXSTP.DB]

Purpose: Provides general data about regional and local STP, such as geographic coordinates, port qualities, and expenses for links to other STPs.

COLUMN HEADINGS	DESCRIPTION
STP_CLLI	STP CLLI code
WCENTER CLLI	CLLI code of Wire Center where STP is located
STATE	Two Character state abbreviation
STP_TYPE	Abbreviation of STP Vendor/Version Modeled
STP_LATITUDE	STP Latitude in degrees
STP_LONGITUDE	STP Longitude in degrees
PORTS_INSVC	Number of STP Ports in Service
PORTCAP	STP Port Capacity
REGIONAL_STP1	Regional STP1 CLLI Code
REGIONAL_STP2	Regional STP2 CLLI Code
CLINKS	Number of C Links
CLINK_DISTANCE	Distance of C Links to the Mate STP
CLINK_EXPENSE	Average Monthly Lease Expense per Link to Mate STP
DLINK_EXPENSE	Average Monthly Lease Expense per Link to Regional STP
DB800LINK_EXP	Average Monthly Lease Expense for Links between Regional STPs and DB800 Regional STPs
LIDBLINK_EXP	Average Monthly Lease Expense for Links between Regional STPs and LIDB Regional STPs
LNPLINK_EXP	Average Monthly Lease Expense for Links between Regional STPs and LNP Regional STPs (not currently used)
AINLINK_EXP	Average Monthly Lease Expense for Links between Regional STPs and AIN Regional STPs (not currently used)

STP Location [XXSTPLOC.DB]

Purpose: For each regional STP, indicates where each pair of service-specific SCPs is located.

COLUMN HEADINGS	DESCRIPTION
REGIONAL_STP1	Regional STP CLLI code
DB800_PriSCP1	CLLI Code of STP colocated with first primary DB800 SCP
DB800_PriSCP2	CLLI Code of STP colocated with second primary DB800 SCP
DB800_SecSCP1	CLLI Code of STP colocated with first secondary DB800 SCP
DB800_SecSCP2	CLLI Code of STP colocated with second secondary DB800 SCP
LIDB_PriSCP1	CLLI Code of STP colocated with first primary LIDB SCP
LIDB_PriSCP2	CLLI Code of STP colocated with second primary LIDB SCP
LIDB_SecSCP1	CLLI Code of STP colocated with first secondary LIDB SCP
LIDB_SecSCP2	CLLI Code of STP colocated with second secondary LIDB SCP
LNP_PriSCP1	CLLI Code of STP colocated with first primary LNP SCP (not currently used).
LNP_PriSCP2	CLLI Code of STP colocated with second primary LNP SCP (not currently used).
LNP_SecSCP1	CLLI Code of STP colocated with first secondary LNP SCP (not currently used).
LNP_SecSCP2	CLLI Code of STP colocated with second secondary LNP SCP (not currently used).
AIN_PriSCP1	CLLI Code of STP colocated with first primary AIN SCP (not currently used).
AIN_PriSCP2	CLLI Code of STP colocated with second primary AIN SCP (not currently used).
AIN_SecSCP1	CLLI Code of STP colocated with first secondary AIN SCP (not currently used).
AIN_SecSCP2	CLLI Code of STP colocated with second secondary AIN SCP (not currently used).

SS7 Investment [XXSS7INV.DB]

Purpose: Provides SS7 hardware and software investment by network component along with estimated labor hours.

COLUMN HEADINGS	DESCRIPTION
NODE_TYPE	Type of SS7 Network Node (e.g. SSP/STP/SCP)
ITEM	Investment Table Item Code (e.g. "SSPPort_HW")
TECHNOLOGY	Type of SSP or STP (e.g. DMS-100, DSC-A32)
ACCOUNT	Part 32 Account Code
HARDWARE	Hardware Investment Value
SOFTWARE	Software Investment Value
ENG_HRS	Engineering Hours
INST_HRS	Installation Hours
TEST_HRS	Installation Testing Hours

SS7 Investment Table Item Descriptions

CODE	DESCRIPTION	IMPACTS
SCPInv	Total Investment per Service Control Point; IBM hardware and Telcordia software investments are identified in respective columns, including vendor installation and engineering. Additional GTE engineering hours are indicated in the table.	DB800 Query, LIDB Query, LNP Query
SCP_SPEC_CKT	The code name is the hardware investment for the Digital Service Unit/Channel Service Unit DSC/CSU) special circuit termination of the A-Link at a SCP. The input table values are engineering labor and installation labor for the special circuit to the SCP.	SS7 Call Setup DB800 Query, LIDB Query, LNP Query
SSP800_SW	Right-to-use (RTU) fees for end office software to support toll-free calling. Includes software for 800, 888, and any other new types of toll-free calling numbers.	DB800 Query (Service Only)
SSPIPort	Investment per port at an end office (SSP), for the initial pair of ports. The investment for the additional ports may differ from the investment for the initial pair of ports because of SS7-related fixed investments, such as frames and spare circuit packs, that are included only in the investments for initial ports. Initial port investments consist of both hardware and software.	SS7 Call Setup
SSPAPort	Hardware investment per additional port at an end office (SSP), beyond the initial pair of ports. The investment for the additional ports may differ from the investment for the initial pair of ports because of SS7-related fixed investments, such as frames and spare circuit packs, that are included only in the investments for initial ports. Depending upon the STP vendor, this item may include software as well as hardware.	SS7 Call Setup, only when more than 2 SSP-STP A links are used.
SSP_SPEC_CKT	The code name is the hardware investment for the Digital Service Unit/Channel Service Unit (DSC/CSU) special circuit termination of the A-Link at an SSP. The input values are engineering labor and installation labor for the special circuit at the SSP	SS7 Call Setup SS7 Call Setup DB800 Query, LIDB Query, LNP Query
STPCommon	Common equipment investment per STP, by vendor and STP type (DMS-STP, DSC-A32, or DSC-Mini). This investment includes all equipment in an STP that is not provisioned on a per port basis. Examples include central processing units, maintenance and test equipment, alarms, and spare circuit packs. Common equipment investment may include both hardware and software.	STP Port, SS7 Call Setup, DB800 Query, LIDB Query, LNP Query
STPPort	Direct investment per port, by vendor and STP type (DMS-STP, DSC-A32, or DSC-Mini). This investment includes only the plug-in cards and equipment having quantities dependent upon number of ports, such as certain shelves and frames. Direct investment per port may include both hardware and software.	STP Port, SS7 Call Setup, DB800 Query, LIDB Query, LNP Query

CODE	DESCRIPTION	IMPACTS
STP_GTTSW	STP software RTU fees required specifically for Global Title Translations (GTT). GTT software allows for appropriate routing of database query messages, such as for toll-free calling or credit card validation, to SCPs. This routing information is in the form of Transaction Capabilities Applications Part (TCAP) messages.	DB800 Query, LIDB Query, LNP Query

Loading Factor [XXLOAD.DB]

Purpose: Provides state-specific minor material loadings and labor rates to convert material investments to total investments. Calculated total investments then include all associated minor material and labor investments.

COLUMN HEADINGS	DESCRIPTION
STATE	Two-Character state abbreviation
HW_FACTOR	Hardware Minor Material/Supply Loading Factor
SW_FACTOR	Software Minor Material/Supply Loading Factor
ENG_RATE	Engineering Labor Rate per Hour (Labor Group (011))
INST_RATE	Installation Labor Rate per Hour (Labor Group (101))
TEST_RATE	Maintenance/Testing Labor Rate per Hour (Labor Group (211))

Network Parameters [XXSS7PAR.DB]

Purpose: Provides SS7 network engineering and utilization data required for cost calculations.

COLUMN HEADINGS	DESCRIPTION
CODE	Network Parameter Code
DESCRIPTION	Description of Network Parameter
VALUE	Network Parameter Value

SS7 Network Parameters Table Item Descriptions

CODE	DESCRIPTION	IMPACTS
OctetCap	Busy hour engineered octet capacity per SS7 link, where an octet is eight bits of signaling information. Links are 56 Kbps circuits engineered at 40 percent maximum occupancy, so the busy hour engineered capacity is 56,000 bits/second * 8 bits/octet * 3600 seconds/hour * 40% = 10,080,000 octets.	SS7 Call Setup, DB800 Query, LIDB Query, LNP Query
CallOctets	Number of signaling octets required to set up an interoffice call. Value from GTE Network Dimensioning document is 100 octets.	SS7 Call Setup
DB8000ctets	Number of signaling octets required for a query to a toll-free calling database. Value from GTE SS7 Network Traffic studies is 94 octets.	DB800 Query
LIDBOctets	Number of signaling octets required for a query to a Line Information Database (LIDB), for services such as credit card validation. Value from GTE SS7 Network Traffic studies is 83 octets.	LIDB Query
LNPOctets	Number of signaling octets required for a query to a Local Number Portability database. Value from GTE SS7 Network Traffic studies is 113 octets.	LNP Query
DB800Util	Average capacity utilization of links to SCPs for toll-free calling databases, including D links between regional STPs and A links between regional STPs and SCPs, as a fraction of parameter *OctetCap*. (User changeable value).	DB800 Query
LIDBUtil	Average capacity utilization of links to SCPs for LIDB queries, including D links between regional STPs and A links between regional STPs and SCPs, as a fraction of parameter *OctetCap*. (User changeable value).	LIDB Query
LNPUtil	Average capacity utilization of links to SCPs for LNP queries, including D links between regional STPs and A links between regional STPs and SCPs, as a fraction of parameter *OctetCap*. (User changeable value).	LNP Query
AlinkUtil	Average capacity utilization of A links between SSPs and local STPs, as a fraction of parameter *OctetCap*. (User changeable value).	SS7 Call Setup
DlinkUtil	Average capacity utilization of D links between local STPs and local STPs, as a fraction of parameter *OctetCap*. (User changeable value).	DB800 Query LIDB Query LNP Query
STPMtceGrowth	STP maintenance and growth factor; investment per port is multiplied by (1 + STPMtceGrowth) to reflect ports reserved for these purposes. Model uses 8 percent of port capacity for growth and 10 percent for maintenance.	STP Port, SS7 Call Setup, DB800 Query, LIDB Query, LNP Query
SCPALinks	Number of A links between a regional STP and an SCP. Model currently uses average value of 16 A links.	DB800 Query, LIDB Query, LNP Query
TestPorts	Number of ports per STP reserved for testing. Model currently uses engineering standard value of 4 ports per STP.	STP Port, SS7 Call Setup, DB800 Query, LIDB Query, LNP Query

CODE	DESCRIPTION	IMPACTS
ISUPOctets	Average number of octets per message for basic interoffice call setup. These messages use ISDN User Part (ISUP) protocol. Model uses 25 octets per message, from Nortel SS7 engineering document. Value is used along with *TCAPOctets*, described below, to estimate TCAP message capacity per STP port.	DB800 Query, LIDB Query, LNP Query
TCAPOctets	Average number of octets per message for database queries. These messages use Transaction Capabilities Applications Protocol (TCAP) protocol. Model uses 80 octets per message, from Nortel SS7 engineering document. Value is used along with *ISUPOctets*, described above, to estimate TCAP message capacity per STP port (GTTMsgCap).	DB800 Query, LIDB Query, LNP Query
GTTMsgCap	Busy Hour capacity per port for TCAP messages. Model assumes that, for each call requiring a database lookup, one TCAP message and five ISUP messages are used. Therefore, TCAP message capacity is estimated, using OctetCap, as 10,080,000 octets per port / ((5 ISUP messages per TCAP message * 25 octets per ISUP message) + 80 octets per TCAP message) = 10,080,000 ÷ 205, or 49,170 TCAP messages per port.	DB800 Query, LIDB Query, LNP Query
STPFill	Fill factor (fraction of capacity) for STP ports. STP port capacity is adjusted by this value for calculating volume-sensitive investment per port. Model currently uses default value of 1 to indicate 100 percent.	STP Port, SS7 Call Setup, DB800 Query, LIDB Query, LNP Query

Data Output Tables (SS7)

The data output tables are listed below. The output tables can be viewed and printed under the menu option **Utilities/Table Viewer/File/Open Table/ACTSS7H.DB.**

SS7 Host Component Output [ACTSS7H]

Purpose: Provides the SS7 network investments by component for host[4] and standalone office. The included components and function of these outputs are defined in the SS7 Output Descriptions table below.

COLUMN HEADINGS	DESCRIPTION
CLLI	The CLLI code for host/standalone office
ACCOUNT	The part 32 account number associated with SS7 investment or cost components
COMPONENT	Describes the investment or unit (e.g., miles, octets, or links) component. See SS7 Output Description Table below for more complete definitions
CATEGORY	Value of 7 identifies the investment or unit component is SS7 related
TOTAL INV	Total incremental component investment or unit value, by CLLI code, for average cost purposes. Investment values are based upon demand by cost driver (e.g., octets, queries, etc.).
VS INV	Incremental component investment or unit values, by CLLI code, for volume-sensitive cost purposes. Unit values in this column are equal to values in the TOTAL INV column. Investment values are based upon usable equipment capacity.
VIS INV	Investment component or unit values, by CLLI code, for volume-insensitive cost purposes. Unit values in this column are equal to values in the TOTAL INV column. Investment values reflect the difference between average and volume-sensitive investments.

[4] ACTSS7R file is populated with the host SS7 component values for each remote.

SS7 Module Output Descriptions (In ACTSS7H and ACTSS7R Tables)

The following components are outputs from the SS7 Investment Module. The values are placed in the ACTSS7H file for host offices. Remote offices use the SS7 links terminating at their respective host offices, so the host values are used for each remote in the ACTSS7R file.

COMPONENT CODE	DESCRIPTION	IMPACTS
ALINK800AERF	Aerial fiber investment required for an A link between a regional STP and 800 database SCP. Value is actually half of the aerial fiber investment per link between the pair of regional STPs. This method is used because each regional STP is colocated with one SCP, requiring no interoffice facilities, while the regional STP does have an interoffice link to the SCP colocated with the other regional STP. Thus, an average of 0 and the full link investment is used.	DB800 Query
ALINK800BURF	Buried fiber investment required for an A link between a regional STP and 800 database SCP. Value is actually half of the buried fiber investment per link between the pair of regional STPs. This method is used because each regional STP is colocated with one SCP, requiring no interoffice facilities, while the regional STP does have an interoffice link to the SCP colocated with the other regional STP. Thus, an average of 0 and the full link investment is used.	DB800 Query
ALINK800CKT	Circuit equipment investment required for an A link between a regional STP and 800 database SCP. Value is actually half of the circuit equipment investment per link between the pair of regional STPs. This method is used because each regional STP is colocated with one SCP, requiring no additional facilities, while the regional STP does have an interoffice link to the SCP colocated with the other regional STP. Thus, an average of the DSU/CSU termination investment and the full link investment is used.	DB800 Query
ALINK800COND	Conduit investment required for an A link between a regional STP and 800 database SCP. Value is actually half of the conduit investment per link between the pair of regional STPs. This method is used because each regional STP is colocated with one SCP, requiring no interoffice facilities, while the regional STP does have an interoffice link to the SCP colocated with the other regional STP. Thus, an average of 0 and the full link investment is used.	DB800 Query
ALINK800POLE	Pole investment required for an A link between a regional STP and 800 database SCP. Value is actually half of the pole investment per link between the pair of regional STPs. This method is used because each regional STP is colocated with one SCP, requiring no interoffice facilities, while the regional STP does have an interoffice link to the SCP colocated with the other regional STP. Thus, an average of 0 and the full link investment is used.	DB800 Query

COMPONENT CODE	DESCRIPTION	IMPACTS
ALINK800UNDF	Underground fiber investment required for an A link between a regional STP and 800 database SCP. Value is actually half of the underground fiber investment per link between the pair of regional STPs. This method is used because each regional STP is colocated with one SCP, requiring no interoffice facilities, while the regional STP does have an interoffice link to the SCP colocated with the other regional STP. Thus, an average of 0 and the full link investment is used.	DB800 Query
ALINKCKT=0	Investment for special termination equipment, beyond the basic equipment required for DS0 terminations. This investment is no longer required because the functions are integrated in channel banks and STP Ports.	SS7 Link - Termination; DB800 Query Service, LIDB Query Service, LNP Query Service.
ALINKDISTANCE	For SSP-STP A links that are on GTE facilities, this value is the sum of the mileages of the two A links in the link pair. If the link pair is leased, this value is 0. If one of the links in the pair is leased, then the mileage for the other link is used. In the calculation of the SS7 Call Setup Cost this total distance is multiplied by the SS7 Link Facility Cost per Mile for a total DS0 investment, which is then converted to investment per octet and to investment per call. This value is also multiplied by ALINKS to obtain the weighting factor for calculating the state-average SS7 Link Facility Cost per Mile.	SS7 Call Setup; DB800 Query Service, LIDB Query Service, LNP Query Service.
ALINKEXP	Total monthly lease expense, where applicable, for the pair of SSP-STP A links, which is used only in the SS7 call setup investment. This value is the sum of the A link lease expenses on each link in the link pair. Generally, a link is leased if it crosses a LATA boundary; otherwise, with a few exceptions, the link is on GTE interoffice facilities.	SS7 Call Setup; DB800 Query Service, LIDB Query Service, LNP Query Service.
ALINKLIDBAERF	Aerial fiber investment required for an A link between a regional STP and LIDB SCP. Value is actually half of the aerial fiber investment per link between the pair of regional STPs. This method is used because each regional STP is colocated with one SCP, requiring no interoffice facilities, while the regional STP does have an interoffice link to the SCP colocated with the other regional STP. Thus, an average of 0 and the full link investment is used.	LIDB Query
ALINKLIDBBURF	Buried fiber investment required for an A link between a regional STP and LIDB SCP. Value is actually half of the buried fiber investment per link between the pair of regional STPs. This method is used because each regional STP is colocated with one SCP, requiring no interoffice facilities, while the regional STP does have an interoffice link to the SCP colocated with the other regional STP. Thus, an average of 0 and the full link investment is used.	LIDB Query

COMPONENT CODE	DESCRIPTION	IMPACTS
ALINKLIDBCKT	Circuit equipment investment required for an A link between a regional STP and LIDB SCP. Value is actually half of the circuit equipment investment per link between the pair of regional STPs. This method is used because each regional STP is colocated with one SCP, requiring no additional facilities, while the regional STP does have an interoffice link to the SCP colocated with the other regional STP. Thus, an average of the DSU/CSU termination investment and the full link investment is used.	LIDB Query
ALINKLIDBCOND	Conduit investment required for an A link between a regional STP and LIDB SCP. Value is actually half of the conduit investment per link between the pair of regional STPs. This method is used because each regional STP is colocated with one SCP, requiring no interoffice facilities, while the regional STP does have an interoffice link to the SCP colocated with the other regional STP. Thus, an average of 0 and the full link investment is used.	LIDB Query
ALINKLIDBPOLE	Pole investment required for an A link between a regional STP and LIDB SCP. Value is actually half of the pole investment per link between the pair of regional STPs. This method is used because each regional STP is colocated with one SCP, requiring no interoffice facilities, while the regional STP does have an interoffice link to the SCP colocated with the other regional STP. Thus, an average of 0 and the full link investment is used.	LIDB Query
ALINKLIDBUNDF	Underground fiber investment required for an A link between a regional STP and LIDB SCP. Value is actually half of the underground fiber investment per link between the pair of regional STPs. This method is used because each regional STP is colocated with one SCP, requiring no interoffice facilities, while the regional STP does have an interoffice link to the SCP colocated with the other regional STP. Thus, an average of 0 and the full link investment is used.	LIDB Query
ALINKOCTETS	Octet demand per SSP-STP A Link. Value is derived as busy hour usable link octet capacity (10,080,000), from SS7 Network Parameters table, times the A link utilization factor, also from the SS7 Network Parameters Table.	SS7 Call Setup
ALINKS	Number of A links between the office (SSP) and its pair of local STPs. Value is used with SSP equipment investment to determine investment per SSP port for call setup cost. ALINKS is also used as the weighting factor for state-average SS7 link termination and STP Port costs. ALINKS is multiplied by ALINKDISTANCE to obtain weighting factor for state-average SS7 Link Facility Cost per Mile.	SS7 Call Setup; SS7 Link – Termination, SS7 Link – Facility Mileage, & STP Port (state-level only)
CALLOCTETS	Number of octets per interoffice call for the signaling portion of call setup, from the SS7 Network Parameters Table. Costs per octet for SSP, SS7 link, and STP port equipment are multiplied by this value to obtain the SS7 cost for call setup.	SS7 Call Setup
DB800OCTETS	Number of octets per query to a database for toll-free calling, from the SS7 Network Parameters Table. Value is multiplied by regional STP and D link costs per octet to obtain those components' contribution to the Cost per DB800 Query.	DB800 Query

COMPONENT CODE	DESCRIPTION	IMPACTS
DLINK800CKT	Circuit equipment investment per D link from the regional STP pair closest to the local STPs to the regional STP pair colocated with the 800 database SCPs. Value is the total investment for circuit equipment at both ends of the link. Model uses the average of the investments for the links to each STP of the primary and secondary regional STP pairs for 800 queries.	DB800 Query (Service Only)
DLINK800EXP	Monthly lease expense for a D link from the regional STP pair closest to the local STPs to the regional STP pair colocated with the 800 database SCPs. Value is an average of the lease expenses to each STP of the primary and secondary regional STP pairs for 800 queries.	DB800 Query (Service Only)
LINK800OCTETS	Average octet demand on D links between pairs of regional STPs for 800 database queries, based on measured SS7 network traffic characteristics. Value is octet capacity per link, from the SS7 Network Parameters Table, times the 800/888 SCP average utilization, also from the Network Parameters Table.	DB800 Query
DLINKAERF	Aerial fiber investment required for a D link between a local STP and a regional STP. Applies only when the local and regional STPs are in the same LATA. Value is the average of investments for links from each local STP of a mated pair to each regional STP in a mated pair.	DB800 Query, LIDB Query, LNP Query (Service Only)
DLINKBURF	Buried fiber investment required for a D link between a local STP and a regional STP. Applies only when the local and regional STPs are in the same LATA. Value is the average of investments for links from each local STP of a mated pair to each regional STP in a mated pair.	DB800 Query, LIDB Query, LNP Query (Service Only)
DLINKCKT	Underground fiber investment required for a D link between a local STP and a regional STP. Applies only when the local and regional STPs are in the same LATA. Value is the average of investments for links from each local STP of a mated pair to each regional STP in a mated pair.	DB800 Query, LIDB Query, LNP Query (Service Only)
DLINKCOND	Conduit investment required for a D link between a local STP and a regional STP. Applies only when the local and regional STPs are in the same LATA. Value is the average of investments for links from each local STP of a mated pair to each regional STP in a mated pair.	DB800 Query, LIDB Query, LNP Query (Service Only)
DLINKEXP	Monthly link lease expense for a D link between a local STP and a regional STP. Applies when the local and regional STPs are in different LATAs. Value is the average of lease expenses for links from each local STP of a mated pair to each regional STP in a mated pair.	DB800 Query, LIDB Query, LNP Query (Service Only)
DLINKOCTETS	Octet demand per D link between local and regional STPs. Value is result of busy hour usable link octet capacity (10,080,000), from SS7 Network Parameters Table, times the D link utilization factor, also from the SS7 Network Parameters Table.	DB800 Query, LIDB Query, LNP Query (Service Only)
DLINKPOLE	Pole investment required for a D link between a local STP and a regional STP. Applies only when the local and regional STPs are in the same LATA. Value is the average of investments for links from each local STP of a mated pair to each regional STP in a mated pair.	DB800 Query, LIDB Query, LNP Query (Service Only)

COMPONENT CODE	DESCRIPTION	IMPACTS
DLINKUNDF	Underground fiber investment required for a D link between a local STP and a regional STP. Applies only when the local and regional STPs are in the same LATA. Value is the average of investments for links from each local STP of a mated pair to each regional STP in a mated pair.	DB800 Query, LIDB Query, LNP Query (Service Only)
LINKLIDBCKT	Circuit equipment investment per D link from the regional STP pair closest to the local STPs to the regional STP pair colocated with the LIDB SCPs. Value is the total investment for circuit equipment at both ends of the link. Model uses the average of the investments for the links to each STP of the primary and secondary regional STP pairs for LIDB queries.	LIDB Query (Service Only)
LINKLIDBEXP	Monthly lease expense for a D link from the regional STP pair closest to the local STPs to the regional STP pair colocated with the LIDB SCPs. Value is an average of the lease expenses to each STP of the primary and secondary regional STP pairs for LIDB queries.	LIDB Query (Service Only)
LINKLIDBOCTETS	Average octet demand on D links between pairs of regional STPs for LIDB queries, based on measured SS7 network traffic characteristics. Value is octet capacity per link, from the SS7 Network Parameters Table, times the LIDB SCP average utilization, also from the Network Parameters Table.	LIDB Query
LIDBOCTETS	Number of octets per query to a Line Information Database, from the SS7 Network Parameters Table. Value is multiplied by regional STP and D link costs per octet to obtain those components' contribution to the Cost per LIDB Query.	LIDB Query
LNPOCTETS	Number of octets per Local Number Portability query, from the SS7 Network Parameters Table. Value is multiplied by regional STP and D link costs per octet to obtain those components' contribution to the Cost per LNP Query.	LNP Query
PORTLIDBAERF	Aerial fiber investment per regional STP port for access to a LIDB database. This investment is for the C links between the mated pair of STPs. The total investment for the C links is divided by the STP port capacity for volume-sensitive costs and by the STP ports in service for average costs.	LIDB Query
PORTLIDBBURF	Buried fiber investment per regional STP port for access to a LIDB database. This investment is for the C links between the mated pair of STPs. The total investment for the C links is divided by the STP port capacity for volume-sensitive costs and by the STP ports in service for average costs.	LIDB Query
PORTLIDBCKT	Circuit equipment investment per regional STP port for access to a LIDB database. This investment is for the C links between the mated pair of STPs. The total investment for the C links is divided by the STP port capacity for volume-sensitive costs and by the STP ports in service for average costs.	LIDB Query
PORTLIDBCOND	Conduit investment per regional STP port for access to a LIDB database. This investment is for the C links between the mated pair of STPs. The total investment for the C links is divided by the STP port capacity for volume-sensitive costs and by the STP ports in service for average costs.	LIDB Query

COMPONENT CODE	DESCRIPTION	IMPACTS
PORTLIDBPOLE	Pole investment per regional STP port for access to a LIDB database. This investment is for the C links between the mated pair of STPs. The total investment for the C links is divided by the STP port capacity for volume-sensitive costs and by the STP ports in service for average costs.	LIDB Query
PORTLIDBSWT	Switching equipment investment per regional STP port for access to a LIDB database. This value consists of the direct hardware and software investment per port, plus the STP common equipment investment. The common equipment investment is divided by the STP port capacity for volume-sensitive costs and by the STP ports in service for average costs.	LIDB Query
PORTLIDBUNDF	Underground fiber investment per regional STP port for access to a LIDB database. This investment is for the C links between the mated pair of STPs. The total investment for the C links is divided by the STP port capacity for volume-sensitive costs and by the STP ports in service for average costs.	LIDB Query
PORT800AERF	Aerial fiber investment per regional STP port for access to an 800 database. This investment is for the C links between the mated pair of STPs. The total investment for the C links is divided by the STP port capacity for volume-sensitive costs and by the STP ports in service for average costs.	DB800 Query
PORT800BURF	Buried fiber investment per regional STP port for access to an 800 database. This investment is for the C links between the mated pair of STPs. The total investment for the C links is divided by the STP port capacity for volume-sensitive costs and by the STP ports in service for average costs.	DB800 Query
PORT800CKT	Circuit equipment investment per regional STP port for access to an 800 database. This investment is for the C links between the mated pair of STPs. The total investment for the C links is divided by the STP port capacity for volume-sensitive costs and by the STP ports in service for average costs.	DB800 Query
PORT800COND	Conduit investment per regional STP port for access to an 800 database. This investment is for the C links between the mated pair of STPs. The total investment for the C links is divided by the STP port capacity for volume-sensitive costs and by the STP ports in service for average costs.	DB800 Query
PORT800POLE	Pole investment per regional STP port for access to an 800 database. This investment is for the C links between the mated pair of STPs. The total investment for the C links is divided by the STP port capacity for volume-sensitive costs and by the STP ports in service for average costs.	DB800 Query
PORT800SWT	Switching equipment investment per regional STP port for access to an 800 database. This value consists of the direct hardware and software investment per port, plus the STP common equipment investment. The common equipment investment is divided by the STP port capacity for volume-sensitive costs and by the STP ports in service for average costs.	DB800 Query

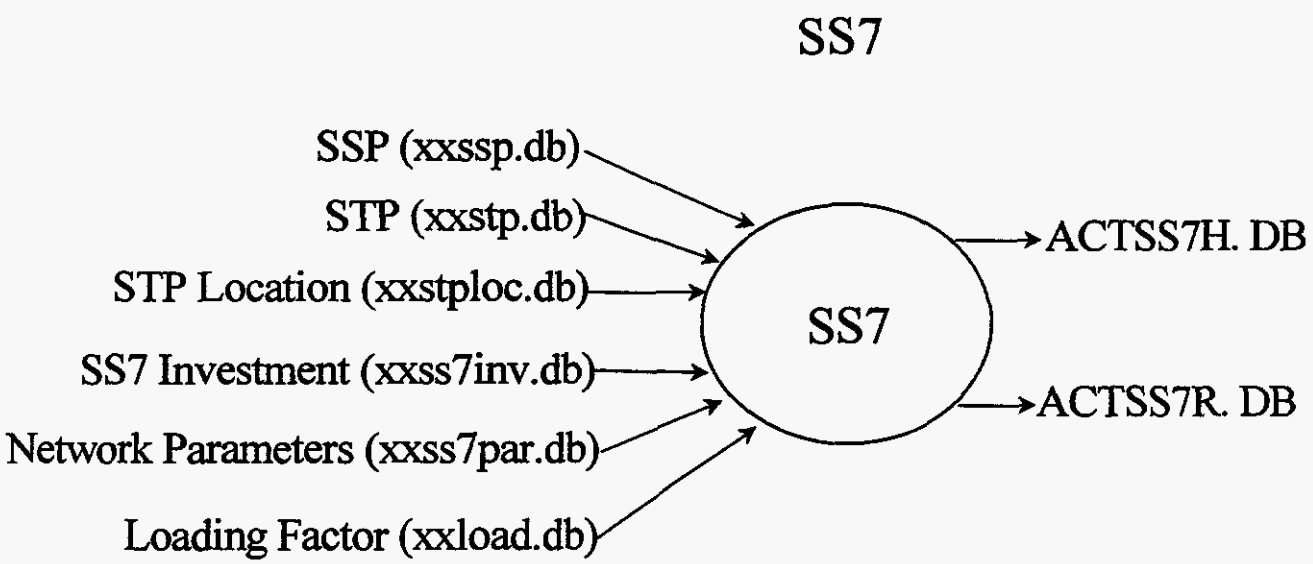
COMPONENT CODE	DESCRIPTION	IMPACTS
PORT800UNDF	Underground fiber investment per regional STP port for access to an 800 database. This investment is for the C links between the mated pair of STPs. The total investment for the C links is divided by the STP port capacity for volume-sensitive costs and by the STP ports in service for average costs.	DB800 Query
RSPORTAERF	Aerial fiber investment per regional STP port. This investment is for the C links between the mated pair of STPs. The total investment for the C links is divided by the STP port capacity for volume-sensitive costs and by the STP ports in service for average costs.	DB800 Query, LIDB Query, LNP Query (Service Only)
RSPORTBURF	Buried fiber investment per regional STP port. This investment is for the C links between the mated pair of STPs. The total investment for the C links is divided by the STP port capacity for volume-sensitive costs and by the STP ports in service for average costs.	DB800 Query, LIDB Query, LNP Query (Service Only)
RSPORTCKT	Circuit equipment investment per regional STP port. This investment is for the C links between the mated pair of STPs. The total investment for the C links is divided by the STP port capacity for volume-sensitive costs and by the STP ports in service for average costs.	DB800 Query, LIDB Query, LNP Query (Service Only)
RSPORTCOND	Conduit investment per regional STP port. This investment is for the C links between the mated pair of STPs. The total investment for the C links is divided by the STP port capacity for volume-sensitive costs and by the STP ports in service for average costs.	DB800 Query, LIDB Query, LNP Query (Service Only)
RSPORTPOLE	Pole investment per regional STP port. This investment is for the C links between the mated pair of STPs. The total investment for the C links is divided by the STP port capacity for volume-sensitive costs and by the STP ports in service for average costs.	DB800 Query, LIDB Query, LNP Query (Service Only)
RSPORTSWT	Switching equipment investment per regional STP port. This value consists of the direct hardware and software investment per port, plus the STP common equipment investment. The common equipment investment is divided by the STP port capacity for volume-sensitive costs and by the STP ports in service for average costs.	DB800 Query, LIDB Query, LNP Query (Service Only)
RSPORTUNDF	Underground fiber investment per regional STP port. This investment is for the C links between the mated pair of STPs. The total investment for the C links is divided by the STP port capacity for volume-sensitive costs and by the STP ports in service for average costs.	DB800 Query, LIDB Query, LNP Query (Service Only)
SCP800	SCP investment per query for toll-free calling, including calls using 800, 888, and other toll-free numbers. Total SCP hardware and software investment is divided by the query capacity for volume-sensitive costs and by estimated query demand for average costs. Query capacity is calculated using the query capacity per SS7 link times 16 average STP-SCP A links per SCP.	DB800 Query

COMPONENT CODE	DESCRIPTION	IMPACTS
SCPLIDB	SCP investment per query to a Line Information Database, for services such as Credit Card Validation. Total SCP hardware and software investment is divided by the query capacity for volume-sensitive costs and by estimated query demand for average costs. Query capacity is calculated using the query capacity per SS7 link times 16 average STP-SCP A links per SCP.	LIDB Query
SSP800PORT	RTU fees for end office software to support access to toll-free calling databases.	DB800 Query (Retail service only)
SSPPORT	End office investment per port for Service Switching Point (SSP) equipment, including hardware and software. Total investment is divided by ALINKS, the number of links terminating at the office.	SS7 Call Setup
STPPORTAERF	Aerial fiber investment per local STP port. This investment is for the C links between the mated pair of STPs. The total investment for the C links is divided by the STP port capacity for volume-sensitive costs and by the STP ports in service for average costs.	STP Port
STPPORTBURF	Buried fiber investment per local STP port. This investment is for the C links between the mated pair of STPs. The total investment for the C links is divided by the STP port capacity for volume-sensitive costs and by the STP ports in service for average costs.	STP Port
STPPORTCKT	Circuit equipment investment per local STP port. This investment is for the C links between the mated pair of STPs. The total investment for the C links is divided by the STP port capacity for volume-sensitive costs and by the STP ports in service for average costs.	STP Port
STPPORTCOND	Conduit investment per local STP port. This investment is for the C links between the mated pair of STPs. The total investment for the C links is divided by the STP port capacity for volume-sensitive costs and by the STP ports in service for average costs.	STP Port
STPPORTEXP	Monthly leased link expense per local STP port. This value, if applicable, is for the C links between the mated pair of STPs. The total lease expense for the C links is divided by the STP port capacity for volume-sensitive costs and by the STP ports in service for average costs. The link lease expense applies only when the mated pair of local STPs are in different LATAs.	STP Port
STPPORTPOLE	Pole investment per local STP port. This investment is for the C links between the mated pair of STPs. The total investment for the C links is divided by the STP port capacity for volume-sensitive costs and by the STP ports in service for average costs.	STP Port
STPPORTSWT	Switching equipment investment per local STP port. This value consists of the direct hardware and software investment per port, plus the STP common equipment investment. The common equipment investment is divided by the STP port capacity for volume-sensitive costs and by the STP ports in service for average costs.	STP Port

COMPONENT CODE	DESCRIPTION	IMPACTS
STPPORTUNDF	Underground fiber investment per local STP port. This investment is for the C links between the mated pair of STPs. The total investment for the C links is divided by the STP port capacity for volume-sensitive costs and by the STP ports in service for average costs.	STP Port
TCAPSWL	Local STP Right-to-Use (RTU) fees for Global Title Translation (GTT) software that provides for routing of database query messages using Transaction Capabilities Application Part (TCAP) protocol. Total investment is divided by TCAP message capacity for volume-sensitive costs and by estimated TCAP message demand for average costs.	DB800 Query, LIDB Query, LNP Query
TCAPSWR	Regional STP Right-to-Use (RTU) fees for Global Title Translation (GTT) software that provides for routing of database query messages using Transaction Capabilities Application Part (TCAP) protocol. Total investment is divided by TCAP message capacity for volume-sensitive costs and by estimated TCAP message demand for average costs.	DB800 Query, LIDB Query, LNP Query

Processing Flow

The SS7 bubble-chart provides a graphic view of the database files that are input to and output of the SS7 Module.



This is a Blank Page

This is a Blank Page



Integrated
Cost Model

Release 4.1

Model Methodology

Expense Module

(Book VI of VII)

GTE Service Corporation

Copyright © GTE 1997 - 2000. All rights reserved.

This is a Blank Page

ICM MODEL METHODOLOGY PACKAGE

- Conceptual Framework — Book I
- Loop Module — Book II
- Switch Module — Book III
- Interoffice Transport Module — Book IV
- SS7 Module — Book V
- Expense Module — Book VI
- Mapping/Report Module — Book VII

This is a Blank Page

Table of Contents

ICM MODEL METHODOLOGY PACKAGE ..... a

ICM EXPENSE MODULE OVERVIEW 1

ICM EXPENSE MODULE METHODOLOGY 2

Capital Cost Factors 2

Operating Expenses 4

Cost Pool Methodology 4

"Activity Based Cost (ABC)" Methodology 13

Shared Costs 17

Common Costs 18

ICM Expense Module – Data Input Tables 19

ICM Expense Module – Output Tables / Reports 20

APPENDIX A: FCC Part 32 Account Code Descriptions – Plant Investment Accounts 21

APPENDIX B: FCC Part 32 Account Code Descriptions – Expense Accounts 26

APPENDIX C1: ICM Expense Methodology – Cost Pools 33

APPENDIX C2: Cost Pool Methodology Roadmap 36

APPENDIX D1: Depreciation & Return Factor Formula and Calculation 37

APPENDIX D2: Composite Income Tax Factor Formula and Calculation 38

APPENDIX E: Data Input Table – Account [XXGTEEXP.DB] 39

APPENDIX F: Data Input Table – Plant Life [XXPLNTLF.DB] 43

APPENDIX G: Data Input Table – Base [XXEXBASE.DB] 45

APPENDIX H: Data Input Table – Property Tax [XXPROPTAX.DB] 47

APPENDIX I: Data Input Table - Non BNF & Activity Based Cost [XXNBNF.DB] 49

APPENDIX J: Output Table – Expense Results [EXPRSLT.DB] 51

APPENDIX K: Output Table – Joint / Shared Cost Table [EXPJOINTSHARED.DB] 53

APPENDIX L: Output Table – Common Cost Table [EXPCCWORKSHEET.DB].....58

APPENDIX M: Output Report – Shared Cost 64

ICM EXPENSE MODULE OVERVIEW

The Expense Module of GTE's Integrated Cost Model (ICM) is designed to develop:

(1) *Capital Cost factors* used to convert the investments developed in the Loop, Switch, Interoffice Transport and SS7 modules into annual expense amounts.

(2) *Operating Expenses* that are associated with the ICM modeled network investment components.

(3) *Shared Costs,* a component of operating expenses, that are in direct support of and can be associated with multiple parts of the network or groups of services, but that can not be directly associated with a specific element of the network, unbundled network element (UNE), product or service.

(4) *Common Costs* that are incurred in support of all products, services and network elements but are not directly attributable to the provision of a specific product, service or unbundled network element.

The *Capital Cost factors* and *Operating Expenses* developed in the Expense Module are available to the Mapping / Report Module in ICM where they are applied to the appropriate modeled investment or product / service to form the monthly costs applicable for each unbundled network element or product and service. Figure UNE.1 provides an example of an ICM output report with the computation of monthly costs for one specific UNE.

Figure UNE.1

UNBUNDLED NETWORK ELEMENTS
NETWORK SERVICES

Network Services	Investment	Depreciation and Return	Composite Income Tax	Property Tax	Maintenance & Support	Marketing	B/C and Directory	TELRIC
Loops Basic								
Basic NAC - 2 Wire	1343.83	205.06	59.46	10.49	46.45	18.78	7.16	28.95
Additional Cost of Unbundling - 2 Wire	148.70	30.71	4.60	1.16	14.29	2.90	0.00	4.47

Capital Costs Operating Expenses

Note: The numbers shown are for presentation purposes only.

ICM EXPENSE MODULE METHODOLOGY

Capital Cost Factors

The capital cost factors developed in the Expense Module of ICM are designed to recover the cost associated with cost of capital, depreciation expense, property taxes and state and federal income taxes. ICM calculates and applies three separate factors to the modeled investment within ICM to determine the amount of necessary costs associated with each investment to be recovered. The capital cost factors developed within ICM are:

(a) Depreciation & Return

(b) Composite Income Tax

(c) Property Tax

(a) The **Depreciation & Return** factor includes both a *return on* and *a return of* invested capital. The *return on* component of the factor is the cost of capital or rate of return. The *return of* component of the depreciation & return factor represents depreciation expense. The formula used to compute the depreciation & return factor is detailed in Appendix D1. A separate depreciation & return factor, based on this formula, is calculated for each plant investment account (Appendix A provides a complete listing of FCC Part 32 plant accounts) utilizing jurisdiction specific Expense Module data inputs related to rate of return, plant depreciable lives and salvage values.

(b) The **Composite Income Tax** factor is used to calculate the amount of state and federal income taxes applicable to the cost developed by ICM. The composite income tax factor

is calculated for each plant investment account category by incorporating statutory state and federal income tax rates, depreciation rates, the weighted average cost of capital, capital structure and cost of debt. The computation for this factor also accounts for differences that may exist between book and tax depreciation methods and is designed to reflect any tax benefits available under the Internal Revenue Service (IRS) Modified Accelerated Capital Recovery System (MACRS) as a result of any such differences. The formula used to compute the composite income tax factor is detailed in Appendix D2.

(c) The **Property Tax** factor utilized in ICM is calculated by taking the ratio of current annual property tax expense to the current gross taxable plant balance.

Operating Expenses

Operating expenses are developed in ICM utilizing a combination of two inter-dependent costing methodologies. These two methodologies allow GTE to analyze and accurately assign its operating expenses based on a cost causative approach. The two methodologies employed in the computation of operating expenses within ICM are the:

(a) Cost Pool methodology, and;

(b) Activity Based Cost (ABC) methodology.

Cost Pool Methodology

The *cost pool* methodology develops forward-looking expenses as a ratio of expenses to investments (E/I). The computation of this ratio (also referred to as the *maintenance & support* factor in the ICM Expense Results output table detailed in Appendix J) is based on a multi-step process as illustrated in Appendix C2. This multi-step process is summarized as follows:

(1) **Base data**: The starting point for the cost pool methodology, GTE's expense and investment data, is based upon the books and records of GTE as summarized in the annual ARMIS Joint Cost Report (43-03), which is filed with the FCC. The ARMIS data is further identified by individual FCC Part 32 accounts (Appendix A & B provide more detailed descriptions) and work centers. Work centers are utilized by GTE as the primary building block for cost centers and are used to capture and report the expense information of a managerial unit or function. It is

at this level of detail that the operating expenses and investments of GTE are analyzed and assigned using the cost pool methodology.

(2) **ARMIS Adjustments:** Certain accounting normalization adjustments are made to the detailed ARMIS data to create a baseline level of expense and investment data for input into ICM. Later in the process certain other types of adjustments are made to produce forward looking operating expenses and to remove costs related to other cost studies such as ABC studies, avoided cost studies, non-recurring studies, and other special studies. Also, in the process, adjustments are developed to bring book plant investments to a current level of cost. All adjustments made to the ARMIS base data are reflected in the detailed expense documentation.

(3) **Adjusted ARMIS Data:** The adjusted baseline data is computed for each FCC Part 32 account as follows:

(1) ARMIS Base Data + (2) ARMIS Adjustments

(4) **Cost Pool Mapping:** The adjusted ARMIS expense and investment data, by FCC Part 32 account, is mapped to one of the twenty-one costs pools that have been established by GTE to classify costs, based on functionality, to various network elements, special studies, lines of business and common costs. The cost pools utilized by GTE in the mapping process are detailed in Appendix C1 and include:

Category	Cost Pool
Network Elements:	
	Cable
	Aerial Non-Metallic Cable
	Aerial Metallic Cable
	Buried Non-Metallic Cable
	Buried Metallic Cable
	Underground Non-Metallic Cable
	Underground Metallic Cable
	Poles
	Conduit
	Transmission
	Switching
	Direct Other
Special Studies:	
	Access
	Information, Originating & Terminating (IOT)
	Operator Services
	Public Communications
Common Costs:	
	Billing & Collection
	Lines of Business:
	Consumer
	Business
	Carrier
	Common

The process of mapping expense and investment data to cost pools utilizes accounting information at the work center level and then, based on the work center functionality is mapped to a cost pool. Investment balances for the respective plant accounts are also mapped to their relative cost pool network function.

(5) Development of Cost Pool Fractions: After the completion of the cost pool mapping process, the costs assigned to individual cost pools for each FCC Part 32 account is converted to a fraction. The cost pool fractions are computed by dividing the expenses assigned to each cost pool by the total account balance for each individual account.

Development of Total Expenses after Adjustments:

(6) Expense Adjustments: In addition to the accounting normalization entries discussed earlier, certain other adjustments are included in ICM to produce forward looking costs and also remove costs that have been included in other studies such as Activity Based Cost (ABC), Non-Recurring Cost (NRC), Avoided Cost and miscellaneous special studies. These adjustments are input into ICM for each individual FCC Part 32 expense account through the use of three factors; Adjust1, Adjust2 and Adjust3. A breakdown of the types of activities associated with each adjustment factor is as follows:

- **Adjust1:** Adjustment factor 1 removes costs from the Adjusted ARMIS data relating to the following cost studies:
    - Avoided Cost (wholesale only)
    - Billing & Collection
    - Non-Recurring Cost
    - Directory Cost

In addition, other adjustments included as part of Adjust1 relate to the removal of Right-to-Use (RTU) expenses related to switch placements or upgrades that are included with the modeled switch investment in ICM as well as any applicable miscellaneous regulatory / normalization entries.

- **Adjust2:** The second expense adjustment factor pertains to the removal of costs from the Adjusted ARMIS data relating to the following Activity Based Cost (ABC) studies:
    - Service Assurance / Repair & Maintenance
    - Sales, Marketing & Advertising

- **Adjust3:** The third expense adjustment factor is used to eliminate the entire Adjusted ARMIS expense amount for a given FCC Part 32 account based on the following items:
    - Non-forward looking costs, such as account 6211 (Analog Switch), 6215 (Electro-mechanical Switch) and 6431 (Aerial Wire).
    - When the combination of Adjust1 plus Adjust2 exceeds 100%

(7) **Adjusted Expense Amount:** The Adjusted Expense Amount for each individual account is computed as follows:

Adjusted ARMIS Expense Data * [[1 – (1 – Adjust1) – (1 – Adjust2)] * Adjust3]

(8) General Support Expenses: In addition to the operating expenses incurred by GTE related to general support expenses (i.e.- FCC Part 32 account 6112 through 6124), GTE also calculates and includes the capital costs associated with General Support Facility assets (i.e.- FCC Part 32 account 2111 through 2124). The capital carrying cost for each general support asset account is computed as follows:

Adjusted ARMIS Investment Data * Adjust1 * Adjust2 * Adjust3 *

(Depreciation & Return factor + Composite Inc Tax factor + Property Tax factor)

The capital carrying cost for each general support asset account is distributed on the basis of the respective cost pool fractions for the associated expense account (i.e.- Account 2124 follows the cost pool assignment of account 6124).

(9) Distribution of Direct Other and Cable Cost Pools: After Total Expenses for each cost pool has been computed, ICM distributes the costs assigned to the Direct Other and Cable cost pools as follows:

- "Direct Other" Cost Pool Assignment – The redistribution of "Direct Other" expenses is based on the relationship between the total for each network cost pool and the sum of all Network cost pools (Cable, Aerial Non-Metallic, Aerial Metallic, Buried Non-Metallic, Buried Metallic, Underground Non-Metallic, Underground Metallic, Poles, Conduit, Transmission, Switch and IOT and is computed as follows:

(Each specific network cost pool amount / sum of all network cost pools)

* Total Direct Other Cost Pool expense

- "Cable" Cost Pool Assignment - Redistribution of shared cable costs such as OSP Construction and Engineering cost to the Cable cost pools (i.e.- Aerial Non-Metallic, Aerial Metallic, Buried Non-Metallic, Buried Metallic, Underground Non-Metallic and Underground Metallic) as follows:

(Each specific cable cost pool amount / sum of all cable cost pools)

* Total Cable Cost Pool expense

(10) **Total Expenses:** After the distribution of the Direct Other and Cable cost pools, total expenses for each cost pool is computed as follows:

Adjusted Expense Amount + General Support Expense

Development of Total Investment after Adjustments:

(11) **Investment Adjustments:** Similar to the expense adjustments discussed above, investment adjustments are input into ICM for each individual FCC Part 32 investment account through the use of three factors; Adjust1, Adjust2 and Adjust3. A breakdown of the types of activities associated with each adjustment factor is as follows:

- **Adjust1:** Investment adjustment 1 applies a C.A. Turner index to each FCC Part 32 investment account in order to adjust the average plant investment balance to current reproduction cost.
- **Adjust2:** The second investment adjustment is not currently used.
- **Adjust3:** The investment adjustment 3 factor is used to:
    - Eliminate the entire Adjusted ARMIS investment amount for a given FCC Part 32 account for non-forward looking costs, such as account 2211 (Analog Switch), 2215 (Electro-mechanical Switch) and 2431 (Aerial Wire).
    - Calibrate the C.A. Turner adjusted ARMIS investment data associated with switch, circuit equipment and outside plant (OSP) accounts to the levels calculated by the Switch, Loop, Interoffice Transport and SS7 modules of ICM.

(12) Total Investment: Total investment is computed as follows:

Adjusted ARMIS Investment Data * Adjust1 * Adjust2 * Adjust3

(13) Computation of Maintenance & Support Factors: After the distribution of the Direct Other and Cable cost pools, ICM computes the Maintenance & Support factors for each network cost pool as follows:

Total Expenses by cost pool / Total Investment by cost pool

The maintenance & support factor for each cost pool is then used in the Mapping / Report Module of ICM where they are applied to the appropriate modeled investment to calculate the monthly maintenance & support cost applicable for each unbundled network element or product and service.

"Activity Based Cost (ABC)" Methodology

The second methodology used in ICM for computing operating expenses is the "Activity Based Cost (ABC)" study approach. The ABC methodology provides GTE a better understanding of the forward-looking nature of certain activities and allows a more precise assignment of the cost of those activities to the causative unbundled network element (UNE), product or service. The ABC methodology incorporates the specific study, identification, definition and costing of activities performed by people and systems in each work center and the identification of the associated costs by cost object, i.e. UNE's, products and services. The costs associated with primary and supporting activities are assigned to UNE's, products and services through the use of various activity drivers. In other words, the cost of a work center is assigned to the cost object that a work center supports. The activity based cost developed from these studies become inputs into ICM and are included in the development of UNE, product or service costs in the Mapping / Report Module of ICM.

Activity based costing studies are prepared for the following four functional activities:

(1) service assurance

(2) sales, marketing & advertising

(3) billing & collection

(4) directory cost

(1) **Service Assurance** is defined as the process of maintaining service levels that consistently meet customer and commission expectations / requirements. The

service assurance cost study identifies work activities related to repairing and maintaining network operations performed by GTE work centers in the Network Services organization. These activities include monitoring the network infrastructure, providing product support, restoring services, and performing preventative maintenance. The methodology utilized in this study requires the identification and costing of those activities associated with GTE personnel and Information Technology / Data Processing (IT/DP) systems in each work center engaged in service assurance activities. Forward-looking costs are incorporated into the service assurance study by utilizing work center management to forecast the impact of any known or planned changes to staffing levels, process modifications, and/or technology enhancements. Activity costs identified through this study are assigned to specific unbundled network elements, products and services based upon their respective consumption of demand units.

The costing principles employed in the service assurance study, as well as the associated detailed supporting documentation, are included under a separate tab in the ICM filing documentation.

(2) **Sales, Marketing & Advertising:** The purpose of this study is to determine the Total Service Long Run Incremental Cost (TSLRIC) and the Total Element Long Run Incremental Costs (TELRIC) for recurring, regulated sales, marketing and advertising (SM&A) expenses associated with products and services. This study

is broken down into three market segment studies based on the type of customer and the services provided. The three studies include:

- Business Market
- Consumer Market
- Carrier Market

The entire sales, marketing & advertising activity based cost study, complete with all supporting documentation for each market, can be found under a separate tab in the ICM filing documentation.

(3) **Billing & Collection:** The Billing & Collection (B&C) activity based cost study is composed of two separate studies. The first study focuses on the ***"End User"*** billing activity that is associated with residential, business and intralata toll customers. This study develops a cost per *access line* and a cost per intralata toll message for all end user billing & collection related activities. The second study is related to ***"Carrier Access Billing Services (CABS)"*** associated with inter-exchange carriers. The CABS study develops the costs per *minute of use* (MOU) for all carrier access billing and collection service related activities.

The entire billing & collection activity based cost study can be found under a separate tab in the ICM filing documentation.

(4) Directory Cost: The Directory study develops the monthly recurring retail and wholesale white page directory costs. This study is focused on three major types of white page directory listings:

- Primary Listings (or main listings), which are basic, single-line listings which are provided to residential and business customers at no incremental fee.
- Premium Listings (or additional listings) include more information than the primary listings. A premium listing occupies no less than two lines in a white page directory. The additional listing is often referred to as a caption.
- Privacy Listings (non-listed or non-published) entail any listing which does not appear in white page directories due to a customer's request for privacy.

Directory costs are assigned to one of the three building blocks listed above. Only the unit costs applicable to primary listings are used in ICM. The complete directory study can be found under a separate tab in the ICM filing documentation

Shared Costs

Shared costs are those costs in direct support of multiple parts of the network or groups of services, but can not be directly associated with a specific element of the network, UNE, product or service. In the development of its shared cost, GTE has identified four categories of shared costs. The four categories include:

(a) ***Outside Plant*** shared cost that are related to, or in direct support of, the parts of the network (poles, aerial, buried and underground cable (copper and fiber) and conduit) that are used to provide UNE's and services (2Wire, R1, etc.) provided by the loop connection.

(b) ***Transmission*** shared cost are those costs that have been identified to be associated with circuit equipment.

(c) ***Switching*** shared cost are those costs that are associated with Central Office equipment and used in the provision of services provided by the central office switching equipment.

(d) ***Network*** shared cost represent the costs that are directly related to the supervision and management of the entire network utilized to provide UNE's, products and services, but *not* a specific UNE, product or service.

Common Costs

Common costs are those costs that can not be specifically assigned to the network or long run incremental costs (TELRIC / TSLRIC) but are incurred by a firm in support of all UNE's, products and services (i.e.- finance, human resources and legal expenses, etc.). The computation of common costs within ICM is determined utilizing the "Cost Pool" methodology outlined in item #2 above. Total common costs are identified in the following cost pools:

- billing & collection
- lines of business (consumer, business and carrier)
- common

Common costs are those that are required to manage and operate the entire company. Common costs are identified in totality for use and assignment for recovery by GTE's pricing group. ICM does not assign common costs to UNE's, TELRIC's or TSLRIC's.

ICM Expense Module – Data Input Tables

There are five database input tables associated with the Expense Module of ICM. These tables include:

Table Description:	Table Name:	Table Overview:
• Account	(XXGTEEXP.DB)	Appendix E
• Plant Life	(XXPLNTLF.DB)	Appendix F
• Base	(XXEXBASE.DB)	Appendix G
• Prop Tax	(XXPROPTAX.DB)	Appendix H
• Non_BNF & Activity Based Costs	(XXNBNF.DB)	Appendix I

The first four tables listed above provide the data required by ICM in the development of the *Capital Cost* factors and the *Maintenance & Support* operating expense factors identified through the "cost pool" methodology. The fifth table contains the cost inputs associated with the service assurance, sales, marketing & advertising, billing & collection and directory costs developed using the "Activity Based Cost (ABC)" methodology.

A detailed description of each table, and its location within ICM, is provided in the Appendixes listed above.

ICM Expense Module – Output Tables / Reports

The information input into the Expense Module database tables discussed in the previous section are processed by ICM and produce the following Output Tables and Reports:

Output Table Description:	Table Name:	Table Overview:
▪ Expense Results Table	(EXPRSLT.DB)	Appendix J
▪ Joint / Shared Cost Table	(EXPJOINTSHARED.DB)	Appendix K
▪ Common Cost Table	(EXPCCWORKSHEET.DB)	Appendix L

Output Report Description:	Report Name:	Report Overview:
▪ Shared Cost Report:		
▪ Page 1	Network Services Shared Cost	Appendix M
▪ Page 2	Summary of Common Costs	Appendix M
▪ Page 3	Comparison Price Fixed Allocator	Appendix M

APPENDIX A: FCC Part 32 Account Code Descriptions – Plant Investment Accounts

ACCOUNT CODE	ACCOUNT TYPE	DESCRIPTION
211100	Land	Includes the original cost of all land held in fee and of easements, and similar rights in land having a term of more than one year used for purposes other than the location of outside plant or externally mounted central office equipment. Also includes special assessments upon land for the construction of public improvements.
211200	Motor Vehicles	Includes the original cost of motor vehicles of the type which are designed and routinely licensed to operate on public streets and highways.
211300	Aircraft	Includes the original cost of aircraft and any associated equipment and furnishings installed as an integral part of the aircraft.
211400	Special Purpose Vehicles	Includes the original cost of special purpose vehicles generally having a service life in excess of two years. It also includes tools and work equipment permanently mounted or forming a part of the vehicle.
211500	Garage Work Equipment	Includes the original cost of tools and equipment used to maintain Motor Vehicles, Aircraft, Special Purpose Vehicles, and Other Work Equipment.
211600	Other Work Equipment	Includes the original cost of power-operated equipment, general-purpose tools and other items of work equipment.
212100	Buildings	Includes the original cost of buildings, and the cost of all permanent fixtures, machinery, appurtenances and appliances installed as a part thereof. It also includes costs incident to the construction and/or purchase of a building and to securing possessions and title.

ACCOUNT CODE	ACCOUNT TYPE	DESCRIPTION
212200	Furniture	Includes the original cost of furniture in offices, storerooms, shops, and all other quarters. It also includes the cost of objects, which possess aesthetic value, are of original or limited edition, and do not have a determinable useful life (Artwork).
212300	Office Equipment	Includes the original cost of office equipment in offices, shops and all other quarters.
212400	General Purpose Computers	Includes the original cost of computer peripheral devices, which are designed to perform general administrative information processing activities.
221100	Analog Electronic Switching	Includes the original cost of stored program control analog circuit-switching and associated equipment. It also includes the cost of remote analog electronic circuit switches.
221200	Digital Electronic Switching	Includes the original cost of Stored Program Control Digital Switches and their associated equipment. Included in this account are digital switches, which utilize either dedicated or non-dedicated circuits. It also includes the cost of remote digital electronic switches.
221500	Electro-Mechanical Switching	Includes the original cost of non-electronic circuit-switching equipment.
222000	Operator Systems	Includes the original cost of those items of equipment used to assist subscribers in utilizing the network, and equipment used in the provision of directory assistance, call intercept, and other assisted call completion activities.
223100	Radio Systems	Includes the original cost of ownership of radio transmitters and receivers.
223200	Circuit Equipment	Includes the original cost of equipment which is used to reduce the number of physical pairs otherwise required to serve a given number of subscribers by utilizing carrier systems, concentration stages or combinations of both. It includes equipment that provides for the simultaneous use of a number of interoffice channels on a single transmission path and includes equipment, which is used for the amplification, modulation, regeneration, circuit patching, balancing or control of signals transmitted over interoffice communications transmission channels.

ACCOUNT CODE	ACCOUNT TYPE	DESCRIPTION
235100	Public Telephone Terminal Equipment	Includes the original cost of coin-less, coin-operated (including public and semi-public), credit card and pay telephones installed for use by the public.
236200	Other Terminal Equipment	Includes the original cost of other Non-CPE terminal equipment not specifically provided for elsewhere and items such as specialized communications equipment provided to meet the needs of the disabled, over-voltage protection equipment, and multiplexing equipment.
241100	Poles	Includes the original cost of poles, cross-arms, guys and other material used in the construction of pole lines and shall include the cost of towers when not associated with buildings.
242110	Metallic Aerial Cable	Includes the original cost of metallic aerial cable and of drop and block wire served by such aerial wire as well as the cost of other material used in the construction of such plant. It also includes the original cost of single or paired conductor cable, wire and other associated material used in constructing a physical path for the transmission of telecommunications signals.
242120	Nonmetallic Aerial Cable	Includes the original cost of nonmetallic aerial cable and of drop and block wire served by such aerial wire as well as the cost of other material used in the construction of such plant. It also includes the original cost of optical fiber cable, and other associated material used in constructing a physical path for the transmission of telecommunications signals.
242210	Underground Cable – Metallic Cable	Includes the original cost of metallic underground cable installed in conduit and of the cost of other material used in the construction of such plant. It also includes the original cost of single or paired conductor cable, wire and other associated material used in constructing a physical path for the transmission of telecommunications signals.
242220	Underground Cable-Nonmetallic Cable	Includes the original cost of nonmetallic underground cable installed in conduit and of the cost of other material used in the construction of such plant. It also includes the original cost of optical fiber cable, and other associated material used in constructing a physical path for the transmission of telecommunications signals.

ACCOUNT CODE	ACCOUNT TYPE	DESCRIPTION
242310	Buried Cable – Metallic	Includes the original cost of buried metallic cable as well as the cost of other material used in the construction of such plant. It also includes the original cost of single or paired conductor cable, wire and other associated material used in constructing a physical path for the transmission of telecommunications signals.
242320	Buried Cable – Nonmetallic	Includes the original cost of buried nonmetallic cable as well as the cost of other material used in the construction of such plant. It also includes the original cost of optical fiber cable, and other associated material used in constructing a physical path for the transmission of telecommunications signals.
242410	Submarine Cable – Metallic	Includes the original cost of submarine metallic cable as well as the cost of other material used in the construction of such plant. It also includes the original cost of single or paired conductor cable, wire and other associated material used in constructing a physical path for the transmission of telecommunications signals.
242420	Submarine Cable – Nonmetallic	Includes the original cost of submarine nonmetallic cable as well as the cost of other material used in the construction of such plant. It also includes the original cost of optical fiber cable, and other associated material used in constructing a physical path for the transmission of telecommunications signals.
242510	Deep Sea Cable – Metallic	Includes the original cost of single or paired conductor cable, wire and other associated material used in constructing a physical path for the transmission of telecommunications signals.
242520	Deep Sea Cable – Non-metallic	Includes the original cost of optical fiber cable and other associated material used in constructing a physical path for the transmission of telecommunications signals.

ACCOUNT CODE	ACCOUNT TYPE	DESCRIPTION
242610	Intra-building Network Cable – Metallic	Includes the original cost of metallic cables and wires located on the company's side of the demarcation point or standard network interface inside subscriber's buildings or between buildings on one customer's same premises. It also includes the original cost of single or paired conductor cable, wire and other associated material used in constructing a physical path for the transmission of telecommunications signals.
242620	Intra-building Network Cable – Non-metallic	Includes the original cost of non-metallic cables and wires located on the company's side of the demarcation point or standard network interface inside subscriber's buildings or between buildings on one customer's same premises. It also includes the original cost of optical fiber cable and other associated material used in constructing a physical path for the transmission of telecommunications signals.
243100	Aerial Wire	Includes the original cost of bare line wire and other material used in the construction of such plant.
244100	Conduit Systems	Includes the original cost of conduit, whether underground, in tunnels or on bridges, which is reusable in place. It also includes the cost of opening trenches and of any paving necessary in the construction of conduit plant.
269030	Intangibles – Network Software – Application & Operating	Network related operating and application system software that is purchased, developed internally or licensed from an external source, including right to use (RTU) fees.

APPENDIX B: FCC Part 32 Account Code Descriptions – Expense Accounts

ACCOUNT CODE	ACCOUNT TYPE	DESCRIPTION
611200	Motor Vehicle Expense	Includes costs of fuel, lubrication, license and inspection fees, washing, repainting, and minor accessories associated with motor vehicles.
611300	Aircraft Expense	Includes such costs as aircraft fuel, flight crews, mechanics and ground crews, licenses and inspection fees, washing, repainting, and minor accessories associated with aircraft.
611400	Special Purpose Vehicles Expense	Includes such costs as fuel, license and inspection fees, washing, repainting, and minor accessories (not operator expense).
611500	Garage Work Equipment Expense	Charged only with costs incurred in connection with the garage work equipment itself (not operator expense).
611600	Other Work Equipment Expense	Charged only with the costs incurred in connection with this work equipment itself (not operator expense).
612100	Land and Building Expense	Includes expenses associated with land and buildings (excluding amortization of leasehold improvements). This account shall also include janitorial service, cleaning supplies, water, sewage, fuel and guard service, and electrical power.
612200	Furniture and Artworks Expense	Includes expenses associated with furniture and artworks such maintenance and repairs, moving and relocating, rental/lease of office equipment, and the cost of insuring furniture and artwork.
612300	Office Equipment Expense	Charged with costs incurred in connection with the office equipment itself, including maintenance and repairs, moving and relocating, rental/lease of office equipment, and the cost of insuring office support equipment.
612400	General Purpose Computers Expense	Includes costs of personnel whose principal job is the physical operation of general-purpose computers and the maintenance of operating systems.

ACCOUNT CODE	ACCOUNT TYPE	DESCRIPTION
621100	Analog Electronic Expense	Includes expense associated with analog electronic switching. Switching equipment consists of equipment located in the central office or elsewhere, which is used to route telephone calls to their destination, and other equipment associated with central office.
621200	Digital Electronic Switching Expense	Includes expenses associated with digital electronic switching. This account includes the cost of repairs, moves and changes, minor purchase, and right-to-use fees.
621500	Electromechanical Switching Expense	Includes expenses associated with electro-mechanical switching. This account includes the cost of repairs, moves and changes, minor purchase, and right-to-use fees.
622000	Operator Systems Expense	Includes expense associated with operator systems equipment. It includes charges related to all types of switchboard equipment except those used solely for recording of calling card telephone numbers in connection with customer charge traffic Operator Number Identification (ONI), and special service switchboards used in conjunction with private line service.
623100	Radio Systems Expense	Includes charges related to equipment associated with radio operations (excluding switchboards).
623200	Circuit Equipment Expense	Includes expenses associated with circuit equipment. It includes charges related to central office equipment other than switching equipment, automatic message recording, operator systems and radio equipment, which is used to derive communication channels or which is used for amplification, modulation, regeneration, testing, balancing or control of signals transmitted over communication transmission channels.
635100	Public Telephone Terminal Equipment Expense	Includes expenses associated with public telephone terminal equipment. It includes the labor and minor material costs of installing, replacing and removing public telephone equipment. It also includes operating leases, rents, and other non-repair items. Beginning in 1997, the FCC deems this activity as non-regulated.

ACCOUNT CODE	ACCOUNT TYPE	DESCRIPTION
636200	Other Terminal Equipment Expense	Includes expenses associated with other terminal equipment. This account also includes costs of repairs, moves and changes, minor purchases, and right-to-use fees. It covers the costs of receiving training on this type of plant as well as operating leases, rents and other non-repair items.
641100	Pole Expense	Includes expenses associated with poles. It includes the cost of relocating guys, cross-arms, cable or wire in connection with such work. It also includes the cost of repairs, moves and changes, rent, training, and minor purchases.
642100	Aerial Cable Expense	Includes charges associated with aerial metallic and optical fiber cable. This account includes the cost of repairs, moves and changes, rent, training, and minor purchases. This account also includes costs related to aerial and optical fiber cable network transmission plant. It includes all charges outside, above ground cables except for intra building Network Cable. It also includes drop and block wire serving aerial cable or aerial wire.
642200	Underground Cable Expense	Includes charges associated with underground cable, both metallic and nonmetallic. It also includes costs of repairs, moves and changes, rent, training, and minor purchases.
642300	Buried Cable	This account shall include expenses associated with buried cable. It includes the cost of repairs, moves and changes, rent, training, and minor purchases.
642400	Submarine Cable Expense	Includes expenses associated with submarine cable, both metallic and nonmetallic. Includes the cost of repairs, moves and changes, rent, training and minor purchases. It also includes charges related to submarine cable network transmission plant.
642500	Deep Sea Cable Expense	Includes charges for both metallic and non-metallic cable, which spans international waters. This account includes costs associated with the costs of repairs, moves and changes, rent, training, and minor purchases related to deep-sea cable.

ACCOUNT CODE	ACCOUNT TYPE	DESCRIPTION
642600	Intra-building Network Cable Expense	Includes charges associated with intra-building network cable, metallic and nonmetallic. Also includes the cost of repairs, moves and changes, rent, training and minor purchases. This account includes charges for customer facility intra-building network cable generally referred to as house cable. It includes indoor risers and distribution cables extending throughout a building or between groups of buildings forming a continuous cable distribution system on one customer's same premises.
643100	Aerial Wire Expense	Includes charges associated with aerial network transmission plant. It includes the cost of repairs, moves and changes, rent, training, and minor purchases.
644100	Conduit System Expense	Includes expenses associated with conduit systems. It includes charges related to conduit and pipes used for underground dips of aerial wire or aerial cable at boulevards, railroads, etc. It includes charges for pole and building connections for conduit systems.
651200	Provisioning Expense	Includes costs incurred in provisioning material and supplies, including office supplies. Also includes receiving and stocking, filling requisitions from stock, monitoring and replenishing stock levels, delivery of material, storage, loading or unloading and administering the reuse or refurbishment of material. Also included are adjustments resulting from the annual or more frequent inventory of material and supplies.
653100	Power Expense	Includes the cost of electrical power used to operate the telecommunications network. It includes the cost of power (excluding labor) for transmitting traffic and operating signals.
653200	Network Administration Expense	Includes costs incurred in network administration. This includes such activities as controlling traffic flow, administering traffic measuring and monitoring devices, assigning equipment and load balancing, collecting and summarizing traffic data, administering trucking, and assigning interoffice facilities and circuit layout work.

ACCOUNT CODE	ACCOUNT TYPE	DESCRIPTION
653300	Testing Expense	Includes costs incurred in testing telecommunications facilities from a testing facility (test desk or other testing system) to determine the condition of plant on either a routine basis, or prior to assignment of the facilities. The costs also include: receiving, recording and analyzing trouble reports; testing to determine the nature and location of reported trouble condition; and dispatching repair persons or otherwise initiating corrective action.
653400	Plant Operations Administration Expense	Includes costs incurred in the general administration of plant operations.
653500	Engineering Expense	Includes costs incurred in the general engineering of the telecommunications plant, which are not directly chargeable to an undertaking or project. This includes developing input to the fundamental planning process, performing preliminary work or advanced planning in connection with potential undertakings, and performing special studies of an engineering nature.
654000	Access Expense	Includes amounts paid by inter-exchange carriers or other exchange carriers to another exchange carrier for the provision of the carrier's carrier access.
656100	Depreciation Expenses – Telecommunications Plant in Service	Includes the depreciation of capitalized costs in Accounts 2112 through 2441 (see above), inclusive.
656200	Depreciation Expense – Property Held for Future Use	This account includes the depreciation expense for capitalized costs included in Account 2002, Property Held for Future Use.
656300	Amortization Expense – Tangible	Includes only the amortization of costs included in Capital Leases (Account 2681) and Leasehold Improvements (Account 2682).
656400	Common Amortization Expense – Intangible	Includes the amortization of costs included in Account 2690, Intangibles.
656500	Amortization – Other	This account includes the amortization of costs recorded to Account 2005, Telecommunications Plant Adjustment.
661100	Product Management	Includes costs incurred in performing administrative activities related to marketing products and services.

ACCOUNT CODE	ACCOUNT TYPE	DESCRIPTION
661200	Sales	Includes costs incurred in selling products and services.
661300	Product Advertising	Includes costs incurred in developing and implementing promotional strategies to stimulate the purchase of products and services.
662100	Call Completing Services	Includes costs incurred in helping customer's place and complete calls, except directory assistance. This includes handling and recording; intercept; quoting rates, time and charges; and all other activities involved in the manual handling of calls.
662200	Number Services	Includes costs incurred in providing customer number and classified listings. This includes preparing or purchasing, compiling, and disseminating those listings through directory assistance or other means.
662300	Customer Services	Includes costs incurred in establishing and servicing customer accounts. Also includes amounts paid by interexchange carriers or other exchange carriers to another exchange carrier for billing and collection services.
671100	Executive	Includes costs incurred in formulating corporate policy and in providing overall administration and management.
671200	Planning	Includes costs incurred in developing and evaluating long-term courses of action for the future operations of the company.
672100	Accounting and Finance	Includes costs incurred in providing accounting and financial services.
672200	External Relations	Includes costs incurred in maintaining relations with government, regulators, other companies and the general public.
672300	Human Resources	Includes costs incurred in performing personnel administration activities.
672400	Information Management	Includes costs incurred in planning, developing, testing, implementing and maintaining databases and application systems for general-purpose computers.
672500	Legal	Includes costs incurred in providing legal services.
672600	Procurement	Includes costs incurred in procuring material and supplies, including office supplies.

ACCOUNT CODE	ACCOUNT TYPE	DESCRIPTION
672700	Research and Development	Includes costs incurred in making planned search or critical investigation aimed at discovery of new knowledge. Also includes translating research findings into a plan or design for a new product or process or for a significant improvement to an existing product or process, whether intended for sale or use.
672800	Other General and Administrative	Includes costs incurred in performing general administrative activities not directly charged to the user, and not provided for in other accounts.
724000	Operating Other Taxes	Includes all taxes, other than federal, state and local income taxes and payroll-related taxes, related to regulated operations applicable to current periods. Among the items included in this account are property, gross receipts, franchise, and capital stock taxes.

APPENDIX C1: ICM Expense Methodology – Cost Pools

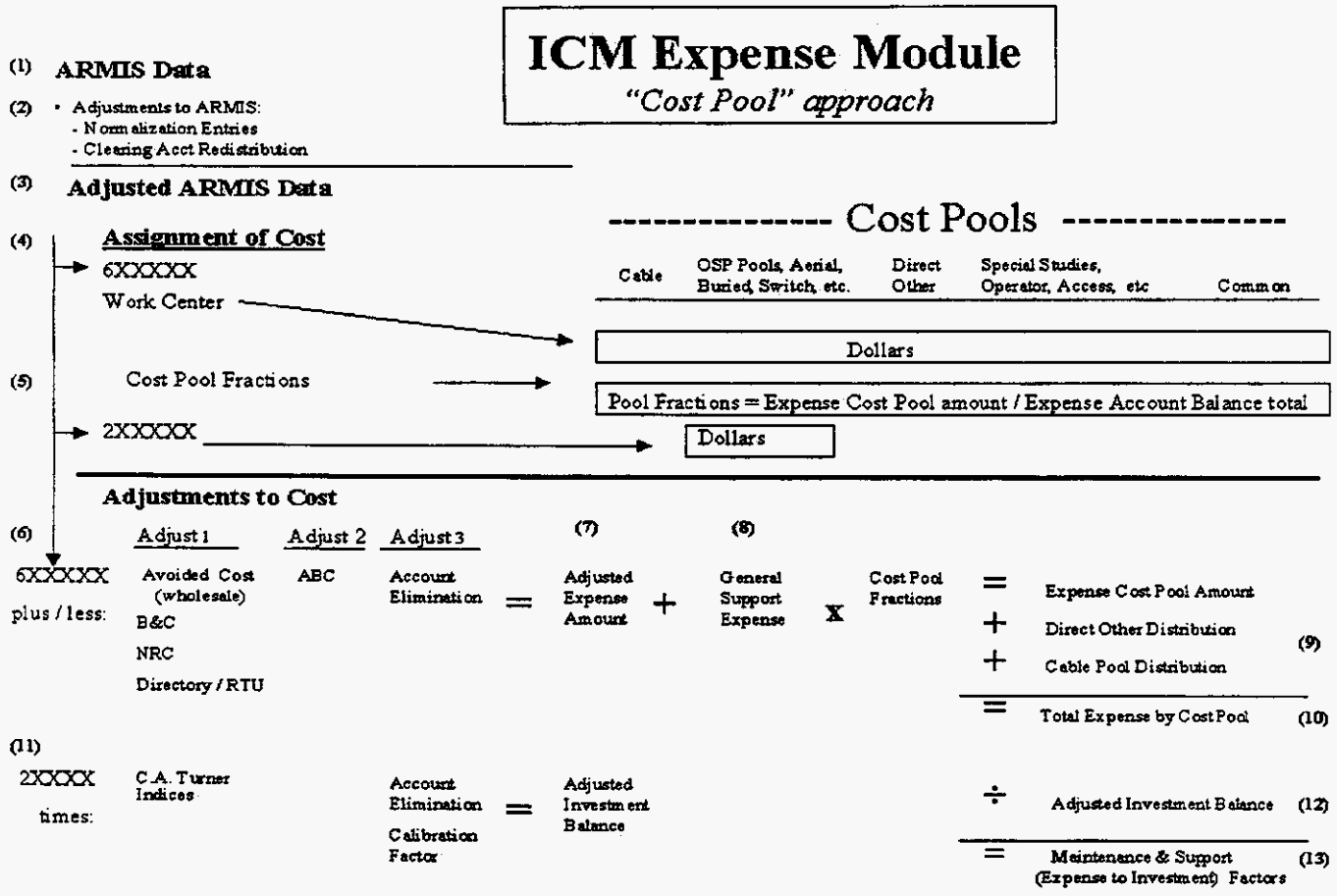
A description of the cost pools and the associated types of costs included in each are listed below.

Cost Pool	Types of Costs included
Cable	Costs related to cable that can not be directly attributable to a specific cable cost pool (i.e.- Aerial Non-Metallic, Aerial Metallic, Buried Non-Metallic, Buried Metallic, Underground Non-Metallic and Underground Metallic). These costs include repair & maintenance (R&M), operational support, capital carrying charges related to general support assets and other costs such as travel, training and rents.
Aerial Non-Metallic	Costs related to Aerial Non-Metallic Cable associated with repair & maintenance (R&M), operational support, capital carrying charges related to general support assets and other costs such as travel, training and rents.
Aerial Metallic	Costs related to Aerial Metallic Cable associated with repair & maintenance (R&M), operational support, capital carrying charges related to general support assets and other costs such as travel, training and rents.
Buried Non-Metallic	Costs related to Buried Nonmetallic Cables associated with repair & maintenance (R&M), operational support, capital carrying charges related to general support assets and other costs such as travel, training and rents.
Buried Metallic	Costs related to Buried Metallic Cables associated with repair & maintenance (R&M), operational support, capital carrying charges related to general support assets and other costs such as travel, training and rents.
Underground Non-Metallic	Costs related to Underground Nonmetallic Cables associated with repair & maintenance (R&M), operational support, capital carrying charges related to general support assets and other costs such as travel, training and rents.

Underground Metallic	Costs related to Underground Metallic Cables associated with repair & maintenance (R&M), operational support, capital carrying charges related to general support assets and other costs such as travel, training and rents.
Poles	Costs related to Poles associated with repair & maintenance (R&M), operational support, capital carrying charges related to general support assets and other costs such as tree trimming, travel, training and rents.
Conduit	Cost related to Conduits associated with repair & maintenance (R&M), operational support, capital carrying charges related to general support assets and other costs such as travel, training and rents.
Transmission	Costs related to Circuit and Transmission equipment associated with repair & maintenance (R&M), operational support, capital carrying charges related to general support assets and other costs such as travel, training and rents.
Switching	Costs related to Switching equipment, COE Engineering, Switch Testing, NOC Centers associated with repair & maintenance (R&M), operational support, capital carrying charges related to general support assets and other costs such as travel, training and rents.
Access	Access charges paid to other carriers
Information, Originating and Terminating (IOT)	Costs related to Information, Originating and Termination equipment associated with repair & maintenance (R&M), operational support, capital carrying charges related to general support assets and other costs such as travel, training and rents.
Direct Other	Costs directly attributed to network operations but not associated with a specific network cost pool (i.e.- Aerial Non-Metallic, Aerial Metallic, Buried Non-Metallic, Buried Metallic, Underground Non-Metallic, Underground Metallic, Poles, Conduit, Transmission, Switching, and IOT).
B & C	Costs related to billing and collection activities.
Operator Services	Costs associated with Operator Services
Public Communication	Costs associated with Public Communication services

Consumer	Costs associated with the Consumer Line of Business
Business	Costs associated with the Business Line of Business
Carrier (CXR)	Costs associated with the Carrier Line of Business
Common	Costs incurred in support of all products, services and network elements but are not directly attributable to the provision of a specific product, service or unbundled network element or a line of business.

APPENDIX C2: Cost Pool Methodology Roadmap



APPENDIX D1: Depreciation & Return Factor Formula and Calculation

Depreciation and Return is a composite of depreciation expense and a return on plant investment.

Depreciation & Return Factor = Annuity (Net Investment Rate, ROR, Plant Life)

where:

i) $$\text{Annuity}(\text{Net Investment Rate}, \text{ROR}, \text{Life}) = \frac{\text{Net Investment Rate} \times \text{ROR} \times (1+\text{ROR})^{Life}}{((1+\text{ROR})^{Life})-1}$$

ii) $$\text{Net InvestmentRate} = 1 - \text{Present Value}(\text{SalvageRate}, \text{ROR}, \text{Life})$$

where:

$$\text{Present Value}(\text{SalvageRate}, \text{ROR}, \text{Life}) = \left(\frac{\text{Salvage Value}}{(1+\text{ROR})^{Life}}\right)$$

iii) ROR = Rate of Return

iv) Life = Economic or usable life of the plant

APPENDIX D2: Composite Income Tax Factor Formula and Calculation

The ***Composite Income Tax*** factor is calculated as follows:

Composite Income Tax Factor =

ITF * (ROR+SFDEPR-SLDEPR) - composite rate * (1+ITF) * (annuity *present value of (MACRS)) - SLDEPR).

Where:

i) ROR = Rate of Return

ii) SFDEPR = Sinking Fund Depreciation =

$$\frac{(1\text{-Salvage Rate}) * ROR}{((1+ROR)^{\wedge life})-1}$$

iii) SLDEPR = Straight Line Depreciation Rate =

$$\frac{1\text{-Salvage}}{Life}$$

iv) $$ITF = \frac{\text{Composite Tax Rate}}{\text{1-Composite Tax Rate}} * \left[1-\left[\frac{\%Debt * Debt\ Rate}{ROR}\right]\right]$$

v) MACRS = Modified Accelerated Capital Recovery System that allows companies to accelerate the depreciation of assets for tax purposes over pre-established class lives.

APPENDIX E: Data Input Table – Account [XXGTEEXP.DB]

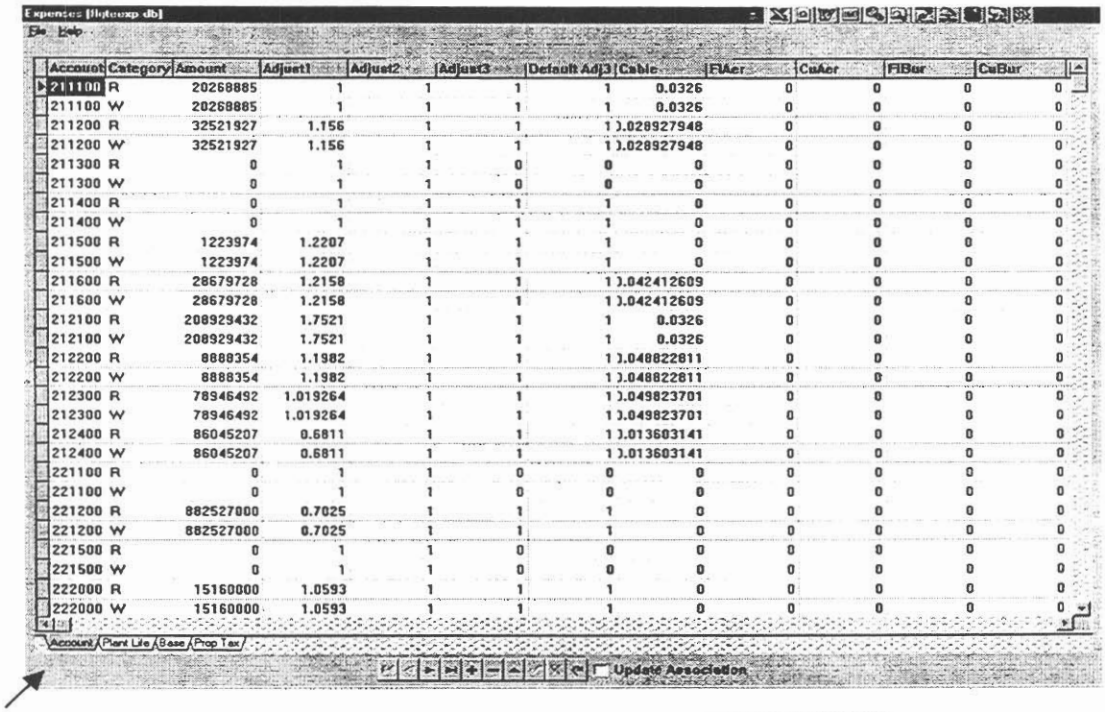
Purpose: Provides expense and investment data, applicable adjustments and the assignment of costs to individual Cost Pools[1] for each respective FCC Part 32 account [2].

How to View On-Line: Using the ICM drop-down menu, select

Table/Expense/Investment Factors/Account tab.

Figure E.1 shown below, illustrates a portion of the data contained within the Account data input table. A complete listing of all data entry fields is found in the section following Figure E.1.

Figure E.1



Expenses [flgteexp.db]

File Help

Account	Category	Amount	Adjust1	Adjust2	Adjust3	Default Adj3	Cable	FlAer	CuAer	FlBur	CuBur
211100	R	20268885	1	1	1	1	0.0326	0	0	0	0
211100	W	20268885	1	1	1	1	0.0326	0	0	0	0
211200	R	32521927	1.156	1	1	1	0.028927948	0	0	0	0
211200	W	32521927	1.156	1	1	1	0.028927948	0	0	0	0
211300	R	0	1	1	0	0	0	0	0	0	0
211300	W	0	1	1	0	0	0	0	0	0	0
211400	R	0	1	1	1	1	0	0	0	0	0
211400	W	0	1	1	1	1	0	0	0	0	0
211500	R	1223974	1.2207	1	1	1	0	0	0	0	0
211500	W	1223974	1.2207	1	1	1	0	0	0	0	0
211600	R	28679728	1.2158	1	1	1	0.042412609	0	0	0	0
211600	W	28679728	1.2158	1	1	1	0.042412609	0	0	0	0
212100	R	208929432	1.7521	1	1	1	0.0326	0	0	0	0
212100	W	208929432	1.7521	1	1	1	0.0326	0	0	0	0
212200	R	8888354	1.1982	1	1	1	0.048822811	0	0	0	0
212200	W	8888354	1.1982	1	1	1	0.048822811	0	0	0	0
212300	R	78946492	1.019264	1	1	1	0.049823701	0	0	0	0
212300	W	78946492	1.019264	1	1	1	0.049823701	0	0	0	0
212400	R	86045207	0.6811	1	1	1	0.013603141	0	0	0	0
212400	W	86045207	0.6811	1	1	1	0.013603141	0	0	0	0
221100	R	0	1	1	0	0	0	0	0	0	0
221100	W	0	1	1	0	0	0	0	0	0	0
221200	R	882527000	0.7025	1	1	1	0	0	0	0	0
221200	W	882527000	0.7025	1	1	1	0	0	0	0	0
221500	R	0	1	1	0	0	0	0	0	0	0
221500	W	0	1	1	0	0	0	0	0	0	0
222000	R	15160000	1.0593	1	1	1	0	0	0	0	0
222000	W	15160000	1.0593	1	1	1	0	0	0	0	0

Account / Plant Life / Base / Prop Tax /

Update Association

Note: The numbers shown in the above illustration are for presentation purposes only.

[1] For a detailed description of the FCC Part 32 account codes, see the Account Code Descriptions for investment accounts found in Appendix A and for expense accounts in Appendix B.

[2] FCC Part 32 Uniform System of Accounts (USOA) for telecommunications companies

COLUMN HEADING	COLUMN DESCRIPTION
ACCOUNT	FCC Part 32 account number
CATEGORY	Indicates Wholesale (W) or Retail (R) information
AMOUNT	Regulated expense and investment data as reported in the FCC ARMIS Joint Cost Report (43-03) adjusted for any applicable normalization entries.
ADJUST1	Factor used to adjust the ARMIS account balances, in the AMOUNT column, for items such as avoided cost (wholesale only), billing and collection, directory cost, right-to-use fees, non-recurring activities and any other applicable adjustments.
ADJUST2	Factor used to adjust ARMIS account balances, in the AMOUNT column, for expenses identified and recovered in the activity based costing methodology (i.e. – service assurance; sales, marketing and advertising, etc.).
ADJUST3	Factor used to adjust the ARMIS account balances, in the AMOUNT column, for the following items: 1) Elimination of non-forward looking costs such as account 2211/6211 (Analog Switch), 2215/6215 (Electromechanical Switch), and 2431/6431 (Aerial Wire). 2) Elimination of account balances in which the combination of ADJUST1 and ADJUST2 exceeds 100% of the ARMIS account balance included in the AMOUNT column. 3) Adjustment to calibrate the investment data associated with Switch, Circuit Equipment, and Outside Plant accounts, to the levels calculated by the Switch, Loop, Interoffice Transport, and SS7 Modules in ICM.
DEFAULT ADJ3	The data contained in this column is identical to the data contained in ADJUST3 except the DEFAULT ADJ3 column does not contain the adjustment used to calibrate the investment associated with the Switch, Circuit Equipment, and Outside Plant accounts. The data in this column is used to reset the data in ADJUST3 when the user elects to run ICM without the affect of investment calibration.
CABLE	Portion of account mapped to Cable cost pool associated with cable costs that cannot be directly attributable to a specific cable cost pool (i.e.- Aerial Non-Metallic, Aerial Metallic, Buried Non-Metallic, Buried Metallic, Underground Non-Metallic and Underground Metallic). These costs include repair & maintenance (R&M), operational support, capital carrying charges related to general support assets and other costs such as travel, training and rents.

COLUMN HEADING	COLUMN DESCRIPTION
FIAER	Portion of account mapped to Aerial Non-Metallic cost pool associated with repair & maintenance (R&M), operational support, capital carrying charges related to general support assets and other costs such as travel, training and rents.
CUAER	Portion of account mapped to Aerial Metallic cost pool associated with repair & maintenance (R&M), operational support, capital carrying charges related to general support assets and other costs such as travel, training and rents.
FIBUR	Portion of account mapped to the Buried Non-Metallic cost pool associated with repair & maintenance (R&M), operational support, capital carrying charges related to general support assets and other costs such as travel, training and rents.
CUBUR	Portion of account mapped to the Buried Metallic cost pool associated with repair & maintenance (R&M), operational support, capital carrying charges related to general support assets and other costs such as travel, training and rents.
FIUND	Portion of account mapped to Underground Non-metallic cost pool associated with repair & maintenance (R&M), operational support, capital carrying charges related to general support assets and other costs such as travel, training and rents.
CUUND	Portion of account mapped to the Underground Metallic cost pool associated with repair & maintenance (R&M), operational support, capital carrying charges related to general support assets and other costs such as travel, training and rents.
POLES	Portion of account mapped to the Poles cost pool associated with repair & maintenance (R&M), operational support, capital carrying charges related to general support assets and other costs such as tree trimming, travel, training and rents.
CONDUIT	Portion of account mapped to the Conduit cost pool associated with repair & maintenance (R&M), operational support, capital carrying charges related to general support assets and other costs such as travel, training and rents.
TRANSMISSION	Portion of account mapped to the Transmission cost pool associated with repair & maintenance (R&M), operational support, capital carrying charges related to general support assets and other costs such as travel, training and rents related to Circuit and Transmission equipment.
SWITCH	Portion of account mapped to the Switch cost pool associated with repair & maintenance (R&M), operational support, capital carrying charges related to general support assets and other costs such as travel, training and rents for Switching equipment, COE Engineering, Switch Testing, NOC Centers.
ACCESS	Portion of account mapped to the Access cost pool associated with Access charges paid to other carriers.

COLUMN HEADING	COLUMN DESCRIPTION
IOT	Portion of account mapped to the IOT cost pool associated with repair & maintenance (R&M), operational support, capital carrying charges related to general support assets and other costs such as travel, training and rents for Information Originating and Terminating equipment.
DIRECT OTHER	Portion of account mapped to the Direct Other cost pool attributed to network operations but not associated with a specific network cost pool (i.e.- Aerial Non-Metallic, Aerial Metallic, Buried Non-Metallic, Buried Metallic, Underground Non-Metallic, Underground Metallic, Poles, Conduit, Transmission, Switching, and IOT).
BILL_PAY	Portion of account mapped to the Billing & Collections cost pool associated with billing and collection activities..
OPER_SERV	Portion of account mapped to the Operator Services cost pool associated with Operator Services costs.
PUBCOM	Portion of account mapped to the Public Communications cost pool associated with Public Communication services costs.
CONSUMER	Portion of account mapped to the Consumer cost pool associated with the Consumer Line of Business.
BUSINESS	Portion of account mapped to the Business cost pool associated with the Business Line of Business.
CXR	Portion of account mapped to the CXR cost pool associated with the Carrier Line of Business.
COMMON	Portion of account mapped to the Common cost pool that relate to costs incurred in support of all products, services and network elements but are not directly attributable to the provision of a specific product, service or unbundled network element or a line of business.

APPENDIX F: Data Input Table – Plant Life [XXPLNTLF.DB]

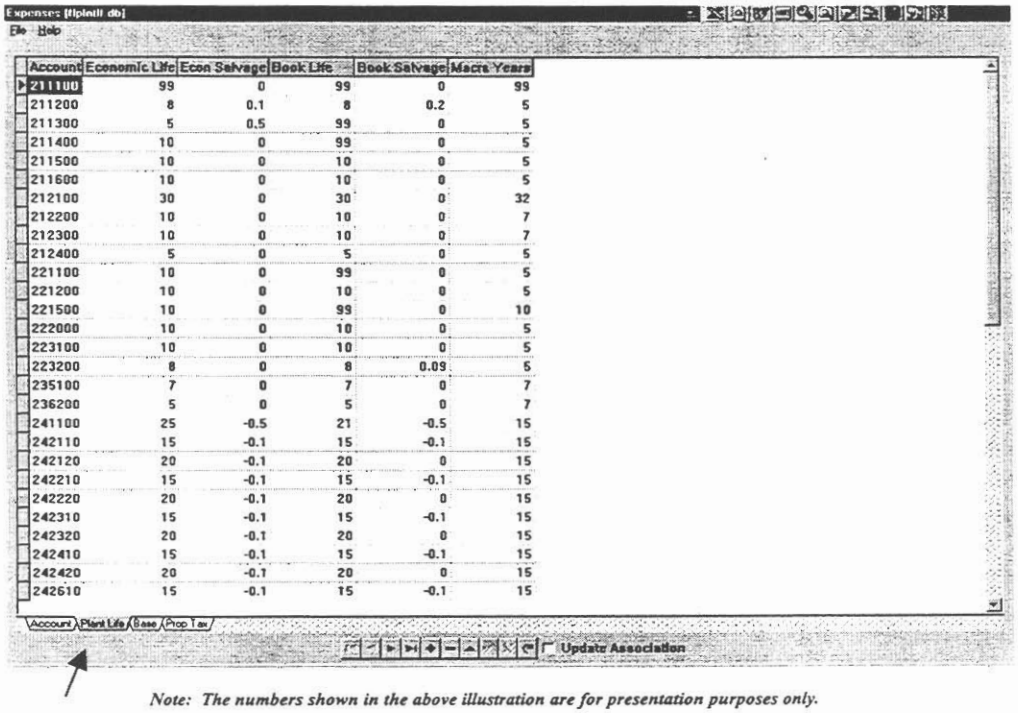
Purpose: Contains the economic, book and MACR's depreciable lives and salvage values for plant investment, by FCC Part 32 account. For a complete listing of all relevant FCC Part 32 investment accounts please refer to Appendix A.

How to View On-Line: Using the ICM drop-down menu, select

Table/Expense/Investment Factors/Plant Life tab.

Figure F.1 shown below illustrates a portion of the data contained within the Plant Life data input table. A detailed listing of all data entry fields is found in the section following Figure F.1.

Figure F.1



Note: The numbers shown in the above illustration are for presentation purposes only.

COLUMN HEADING	COLUMN DESCRIPTION
ACCOUNT	FCC Part 32 account number for investment related accounts.
ECONOMIC LIFE	The economic life associated with plant investment used in the calculation of capital recovery and composite income tax within ICM.
ECON SALVAGE	The economic life salvage percentage associated with plant investment used in the calculation of capital recovery and composite income tax within ICM.
BOOK LIFE	PUC/PSC mandated depreciable life associated with plant investment used in the calculation of capital recovery and composite income tax within ICM.
BOOK SALVAGE	PUC/PSC mandated depreciable life salvage percentage associated with plant investment used in the calculation of capital recovery and composite income tax within ICM.
MACRS YEARS	Depreciable Life according to the Modified Accelerated Capital Recovery System mandated by the IRS.

APPENDIX G: Data Input Table – Base [XXEXBASE.DB]

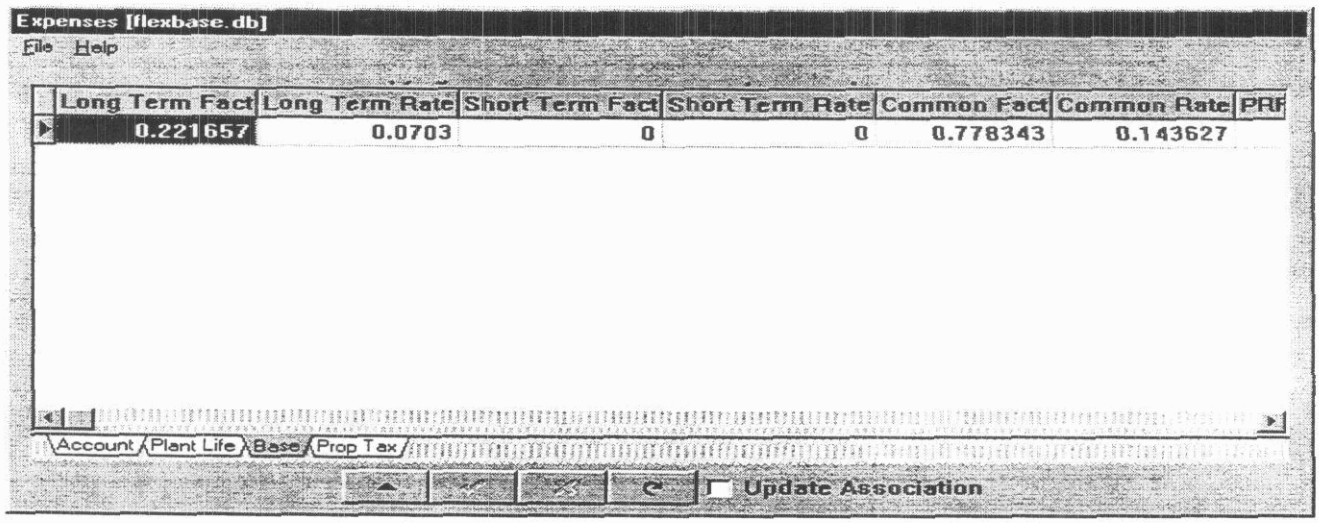
Purpose: This table contains the capital structure (i.e., fraction of capital composed of short-term debt, long-term debt, common stock, and preferred stock), cost of capital, statutory state and federal income tax rates, property tax rate and annual regulated revenues.

How to View On-Line: Use the ICM drop-down menu and select

Table/Expense/Investment Factors/Base tab.

Figure G.1 shown below, illustrates a portion of the data contained within the Base data input table. A detailed listing of all data entry fields is found in the section following Figure G.1.

Figure G.1



Note: The numbers shown in the above illustration are for presentation purposes only.

COLUMN HEADING	COLUMN DESCRIPTION
LONG TERM FACT	The portion of the company's capitalization that is long-term debt.
LONG TERM RATE	Interest rate the company expects to pay on long-term debt
SHORT TERM FACT	The portion of the company's capitalization that is short-term debt.
SHORT TERM RATE	Interest rate the company expects to pay on short-term debt.
COMMON FACT	The portion of the company's capital structure that includes common equity.
COMMON RATE	Cost of financing with common equity.
PRFD FACT	Portion of the company's capital structure that is preferred stock.
PRFD RATE	Expected cost to the company of its preferred stock
RATE RETURN	Return on capital; this field is calculated within ICM as follows: (LONG TERM FACT * LONG TERM RATE) + (SHORT TERM FACT * SHORT TERM RATE) + (COMMON FACT * COMMON RATE) + (PRFD FACT * PRFD RATE).
STATE RATE	Statutory State Income Tax rate applicable for each individual state.
FEDERAL RATE	Statutory Federal Income Tax rate.
PROPERTY TAX	Property tax rate applicable for each individual state. This rate is calculated by taking the ratio of current annual property tax expense to the current gross taxable plant balance.
TOTAL REVENUES	Total regulated revenues, reported in the ARMIS Joint Cost report (43-03), used in the computation of the common cost factors.
DATA YEAR	Year applicable to the ARMIS Joint Cost report (43-03) associated with the "total revenues" shown above.

APPENDIX H: Data Input Table – Property Tax [XXPROPTAX.DB]

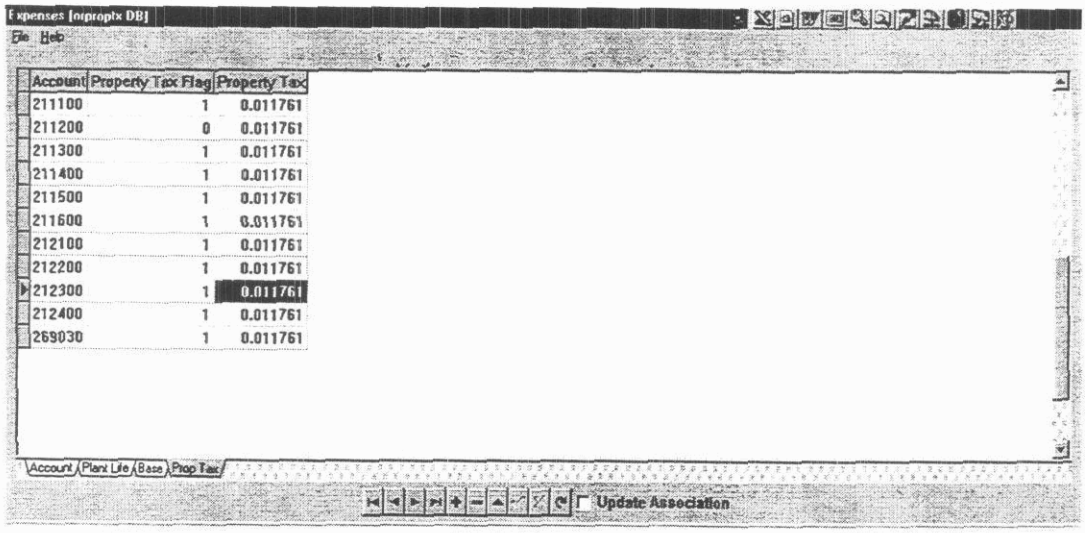
Purpose: Identifies whether property tax is applicable, based on state requirements, for general support investments (i.e. – FCC Part 32 account 2111 through 2124) and internal use software (i.e. -- FCC Part 32 account 2690).

How to View On-Line: Use the ICM drop-down menu and select

Table/Expense/Investment Factors/PropTax tab.

Figure H.1 shown below illustrates the data contained within the Property Tax data input table. A detailed listing of all data entry fields is found in the section following Figure H.1.

Figure H.1



Note: The numbers shown in the above illustration are for presentation purposes only.

COLUMN HEADING	DESCRIPTION
ACCOUNT	FCC Part 32 account number for general support investments (i.e. – account 211100 through 212400) and internal use software (i.e. – account 269030).
PROPERTY TAX FLAG	Identifies the general support investment and internal use software accounts that should have property taxes applied to them. An account in which a "1" appears in the property tax flag field indicates that property tax is applicable to that particular investment account and a "0" indicates property tax is not applicable.
PROPERTY TAX	The property tax rate developed for a given state. This value is calculated based on Total Actual/Estimated Property Taxes divided by Gross Taxable Investment.

APPENDIX I: Data Input Table - Non BNF & Activity Based Cost [XXNBNF.DB]

Purpose: This input table contains the Non-BNF and Activity Based Cost components for expenses identified in the activity based cost studies.

How to View On-Line: Use the ICM drop-down menu and select

Table/Expense/ Non BNF & Activity Based Costs

Figure I.1 shown below illustrates the data contained within the Non BNF & Activity Based Cost data input table. A detailed listing of all data entry fields is found in the section following Figure I.1.

Figure I.1

Non BNF and Activity Based Costs [flnbnf.db]

File Help

Item	Description	Average	VS	VIS
ADCOIN	ADV Coin	0.009036	0.009036	0.009036
ADCUSTOM	ADV Vertical Services	0.015713	0.015713	0.015713
ADISDNB	ADV ISDN BRI	0.009036	0.009036	0.009036
ADISDNP	ADV ISDN PRI	0.009036	0.009036	0.009036
ADMEABUS	ADV Measured Business	0.009036	0.009036	0.009036
ADMEACON	ADV Measured Consume	0.016535	0.016535	0.016535
ADMESS	ADV Message Toll	0.01521	0.01521	0.01521
ADOWN	ADV Cust Owned Pay	0.009036	0.009036	0.009036

Update Association

Note: The numbers shown in the above illustration are for presentation purposes only.

COLUMN HEADINGS	COLUMN DESCRIPTION
ITEM	Item variable name that denotes the Non-BNF or Activity Based Cost component and associated activity.
DESCRIPTION	Description of the Non-BNF or Activity Based Cost component and associated activity. ADV Advertising MKT Marketing SAL Sales BC Billing and Collection DIR Directories AC Access MR Service Assurance – Services All other variables Service Assurance - UNEs
AVERAGE	Sum of the Volume Sensitive and Volume Insensitive components
VS	Volume Sensitive. This is the cost which increases or decreases with the level of output of a product produced at a positive level of output.
VIS	Volume Insensitive. Fixed cost that does not change based on the level of output.

APPENDIX J: Output Table – Expense Results [EXPRSLT.DB]

Purpose: This table contains the ICM computed depreciation and return, composite income tax, maintenance & support and property tax factors for each respective FCC Part 32 investment account [3].

How to View On-Line: Use the ICM drop-down menu and select

Utilities/Table Viewer/File/Open/Exprslt.db

Figure J.1 shown below, illustrates a portion of the data contained within the Expense Results output table. A complete listing of all data entry fields is found in the section following Figure J.1.

Figure J.1

ICM Report Viewer

File Help

December 27, 1999 15:15 File: "Exprslt"

Integrated Cost Model (ICM)
Copyright 1997 - 1999 GTE Telephone Operations
Exprslt.db

	Account	Depreciation and Return	Composite Income Tax	Maintenance and Support	Property Tax
1	211100	0.127374449162843	0.0706821470780754		0.011761
2	211200	0.198602154718842	0.0250839383405969		
3	211300	0.204934163428502	-0.00501305582282477		0.011761
4	211400	0.182359187383316	0.0260327634183542		0.011761
5	211500	0.182359187383316	0.0260327634183542		0.011761
6	211600	0.182359187383316	0.0260327634183542		0.011761
7	212100	0.130963733020612	0.0557773474657525		0.011761
8	212200	0.182359187383316	0.0324439520568217		0.011761
9	212300	0.182359187383316	0.0324439520568217		0.011761
10	212400	0.28249476969696004	0.0424554684191121		0.011761
11	221100	0.182359187383316	0.0260327634183542		
12	221200	0.182359187383316	0.0260327634183542	0.246212665164702	0.011761
13	221500	0.182359187383316	0.0405373291823906		
14	222000	0.182359187383316	0.0260327634183542		
15	223100	0.182359187383316	0.0260327634183542	0.0374829503242693	0.011761
16	223200	0.206516443336638	0.0300359805511013	0.0374829503242693	0.011761
17	235100	0.2242654590337759	0.0408376234975104		
18	236200	0.28249476969696004	0.0523871164149684		
19	241100	0.1374131430261911	0.0426194556793907	0.225391694467384	0.011761
20	242110	0.155174820278996	0.0480559175525695	0.00711347865838651	0.011761
21	242120	0.141385919080635	0.0429520731818725	0.00159886079626869	0.011761
22	242210	0.155174820278996	0.0480559175525695	0.00397453469842881	0.011761

1 of 1

Note: The numbers shown in the above illustration are for presentation purposes only.

[3] FCC Part 32 Uniform System of Accounts (USOA) for telecommunications companies; See Appendix A.

COLUMN HEADING	COLUMN DESCRIPTION
ACCOUNT	FCC Part 32 investment account number
DEPRECIATION & RETURN	Factor developed within ICM that includes the "return on" and "return of" invested capital.
COMPOSITE INCOME TAX	Factor calculated in ICM that represents a composite rate of state and federal income taxes.
MAINTENANCE & SUPPORT	Factor developed in ICM utilizing the "Cost Pool" methodology in the development of forward-looking expenses as a ratio of expense to investment.
PROPERTY TAX	Represents the property tax rate developed for a given state based on Total Actual/Estimated Property Taxes divided by Gross Taxable Investment.

APPENDIX K: Output Table – Joint / Shared Cost Table [EXPJOINTSHARED.DB]

Purpose: This table contains a summary of the results of the "Cost Pool" mapping process in the development of forward looking expenses based on the ratio of expenses to investment. This table also provides a summary of Total Costs broken down between direct cost, shared cost and non-network costs.

How to View On-Line: Use the ICM drop-down menu and select

Utilities/Table Viewer/File/Open/Expjointshared.db

Figure K.1 shown below, illustrates a portion of the data contained within the Joint / Shared Cost output table. A complete listing of all data entry fields is found in the section following Figure K.1.

Figure K.1

Query Table: ExpJointShared.DB

File Help

Logical | Select Field | Op | Field Value | Execute | Clear | Sort

Account	Category	Total	Cable	Aerial Non-Metallic	Aerial Metallic	Buried Non-Metallic	Buried M
6722	W	$8,263,101.00	1601	0	0	0	
6723	W	$10,082,742.00	101896	0	0	0	
6724	W	$36,554,415.00	8634	0	0	0	
6725	W	$3,595,109.00	0	0	0	0	
6726	W	$1,786,324.00	0	0	0	0	
6727	W	$3,557,336.00	0	0	0	0	
6728	W	$29,629,150.00	2093194	0	0	0	
7240	W	$27,090,165.00	0	0	0	0	
A) Subtotal	W	$440,421,181.00	9979657	488	588576	8105	702
B) Total after redistribution	W	$440,421,181.00		1932	2332191	32114	2783
C) Total Investment	W		0	1208621	327855201	9989627	170396
D) Ratios (B/C)	W		0	0.0015988007962686 9	0.007113478658386 51	0.003214711795989 27	35536384
E)	W						
F) Expense Summary:	W						
G) Direct Cost	W	$44,415,409.00	0	488	588576	8105	702
H) Shared Cost	W	$200,342,377.00					
I) Non-Network Cost	W	$195,663,395.00					
I) Summary Total	W	$440,421,181.00					

Result set record count: 73

Current Path: C:\ICM\database\

Note: The numbers shown in the above illustration are for presentation purposes only.

COLUMN HEADING	COLUMN DESCRIPTION
ACCOUNT	FCC Part 32 account number
CATEGORY	Indicates whether information is related to Wholesale (W) or Retail (R) products and services.
TOTAL	Based on regulated expense and investment data as reported in the FCC ARMIS Joint Cost Report (43-03) adjusted for normalization entries and the effect of Adjust1, Adjust2 and Adjust3.
CABLE	Portion of account mapped to Cable cost pool associated with repair & maintenance (R&M), operational support, capital carrying charges related to general support assets and other costs such as travel, training and rents
AERIAL NON-METALLIC	Portion of account mapped to Aerial Non-Metallic cost pool associated with repair & maintenance (R&M), operational support, capital carrying charges related to general support assets and other costs such as travel, training and rents.
AERIAL METALLIC	Portion of account mapped to the Aerial Metallic cost pool associated with repair & maintenance (R&M), operational support, capital carrying charges related to general support assets and other costs such as travel, training and rents.
BURIED NON-METALLIC	Portion of account mapped to the Buried Non-Metallic cost pool associated with repair & maintenance (R&M), operational support, capital carrying charges related to general support assets and other costs such as travel, training and rents.
BURIED METALLIC	Portion of account mapped to the Buried Metallic cost pool associated with repair & maintenance (R&M), operational support, capital carrying charges related to general support assets and other costs such as travel, training and rents.
UNDERGROUND NON-METALLIC	Portion of account mapped to Underground Non-metallic cost pool associated with repair & maintenance (R&M), operational support, capital carrying charges related to general support assets and other costs such as travel, training and rents.
UNDERGROUND METALLIC	Portion of account mapped to the Underground Metallic cost pool associated with repair & maintenance (R&M), operational support, capital carrying charges related to general support assets and other costs such as travel, training and rents.
POLES	Portion of account mapped to the Poles cost pool associated with repair & maintenance (R&M), operational support, capital carrying charges related to general support assets and other costs such as tree trimming, travel, training and rents.
CONDUIT	Portion of account mapped to the Conduit cost pool associated with repair & maintenance (R&M), operational support, capital carrying charges related to general support assets and other costs such as travel, training and rents.

COLUMN HEADING	COLUMN DESCRIPTION
TRANSMISSION	Portion of account mapped to the Transmission cost pool associated with repair & maintenance (R&M), operational support, capital carrying charges related to general support assets and other costs such as travel, training and rents related to Circuit and Transmission equipment.
SWITCH	Portion of account mapped to the Switch cost pool associated with repair & maintenance (R&M), operational support, capital carrying charges related to general support assets and other costs such as travel, training and rents for Switching equipment, COE Engineering, Switch Testing, NOC Centers.
IOT	Portion of account mapped to the IOT cost pool associated with repair & maintenance (R&M), operational support, capital carrying charges related to general support assets and other costs such as travel, training and rents for Information Originating and Terminating equipment.
DIRECT OTHER	Portion of account mapped to the Direct Other cost pool attributed to network operations but not associated with a specific network cost pool (i.e.- Aerial Non-Metallic, Aerial Metallic, Buried Non-Metallic, Buried Metallic, Underground Non-Metallic, Underground Metallic, Poles, Conduit, Transmission, Switching, and IOT).
ACCESS	Portion of account mapped to the Access cost pool associated with Access charges paid to other carriers
B AND C	Portion of account mapped to the Billing & Collection cost pool associated with billing & collection activities.
OPER	Portion of account mapped to the Operator Services cost pool associated with Operator Services costs.
PUB COMM	Portion of account mapped to the Public Communications cost pool associated with Public Communication services costs.
CONSUMER	Portion of account mapped to the Consumer cost pool associated with the Consumer Line of Business.
BUSINESS	Portion of account mapped to the Business cost pool associated with the Business Line of Business
CXR	Portion of account mapped to the CXR cost pool associated with the Carrier Line of Business.
COMMON	Portion of account Total mapped to the Common cost pool associated with costs incurred in support of all products, services and network elements but are not directly attributable to the provision of a specific product, service or unbundled network element or a line of business.

In addition to the information for each FCC Part 32 expense account highlighted above, this database table also contains selected summary information computed within ICM. This information is found in the last ten rows of the table and includes the following data:

ROW HEADING	ROW DESCRIPTION
(A) Subtotal	The sum of all general support costs (FCC Part 32 accounts 2111 through 2124) and operating expenses (accounts 6112 through 6728 and 7240) for each individual cost pool listed above
(B) (B1) Total after Redistribution	Represents the computation of either "total expenses" or "direct expenses" used in the computation of the Maintenance & Support operating expense factors identified in row 'D' described below. The expense amount computed and displayed on this row is either total or direct expense based on whether the user elects to "include shared cost" or "not include shared cost" before running ICM. The data computed in this field represents: ▪ When the user elects to *include* shared cost (see Appendix M, section A titled "Select Report Parameters"), the expense amount appearing on this line represents the costs identified in item "(A) Subtotal" adjusted to include: ❑ The distribution of "Direct Other" expenses based on the relationship between the total for each network cost pool and the sum of all Network cost pools (Cable, Aerial Non-Metallic, Aerial Metallic, Buried Non-Metallic, Buried Metallic, Underground Non-Metallic, Underground Metallic, Poles, Conduit, Transmission, Switch and IOT). ❑ The distribution of shared cable costs to the Cable Cost Pools (i.e.- Aerial Non-Metallic, Aerial Metallic, Buried Non-Metallic, Buried Metallic, Underground Non-Metallic and Underground Metallic).
(B2) Direct Costs	When the user elects to ***not*** *include* shared cost, the expense amount, appearing on this line represents the costs identified as "Direct Costs". Direct costs represent plant specific expenses recorded to FCC Part 32 accounts 6211 through 6441 that are assigned to the network cost pools through the "Cost Pool" mapping process.
(C) Total Investment	The sum of plant investment for all FCC Part 32 accounts 2211 through 2441 and account 26903x for each network cost pool.

ROW HEADING	ROW DESCRIPTION
(D) Ratios (B/C)	Represents the computation of the Maintenance & Support factor in ICM. The formula utilized is different based on whether the user selected to "include shared cost" or "not include shared cost" before running ICM. The two formulas utilized in ICM include: ▪ When shared costs are *included* as part of total operating expenses, the formula for computing the Maintenance & Support operating expense factor for each network cost pool is as follows: (B1)Total after Redistribution / (C) Total Investment ▪ When shared costs are ***not included*** as part of total operating expenses, the formula for computing the Maintenance & Support operating expense factor for each network cost pool is as follows: (B2) Direct Cost / (C) Total Investment
(E)	Not currently used
(F) Expense Summary:	Title field only
(G) Direct Cost	Represents plant specific expenses recorded to FCC Part 32 accounts 6211 through 6441 that are assigned to the network cost pools through the "Cost Pool" mapping process.
(H) Shared Cost	Those costs in direct support of specific parts of the network or groups of services that are recorded to all FCC Part 32 accounts, other than accounts 6211 through 6441, and mapped to the network cost pools through the "Cost Pool" mapping process.
(I) Non-Network Cost	Represents expenses assigned through the "Cost Pool" mapping process to the non-network cost pools such as Access, Operator, Public Communications, Billing and Collection, Lines of Business (Consumer, Business and CXR) and Common.
(J) Summary Total	Sum of item G (Direct Cost) + item H (Shared Cost) + item I (Non-Network Cost)

APPENDIX L: Output Table – Common Cost Table [EXPCCWORKSHEET.DB]

Purpose: This table contains a detailed development of the components used in the computation of Common Costs within ICM.

How to View On-Line: Use the ICM drop-down menu and select

Utilities/Table Viewer/File/Open/Expccworksheet.db

Figure L.1 shown below, illustrates a portion of the data contained within the Common Cost output table. A complete listing of all data entry fields is found in the section following Figure L.1.

Figure L.1

Query Table: ExpCCWorksheet.DB

File Help

Logical | Select Field | Op | Field Value | Execute | Clear | Sort

Account	Category	Amount	Adjust1	Adjust2	Adjust3	New Amount	%BC	%Consumer
2111	W	$20,268,885.00	1	1	1	$20,268,885.00	0.0014	
2112	W	$32,521,927.00	1.156	1	1	$37,595,347.61	0	
2113	W	$0.00	1	1	0	$0.00	0	
2114	W	$0.00	1	1	1	$0.00	0	
2115	W	$1,223,974.00	1.2207	1	1	$1,494,105.06	0	
2116	W	$28,679,728.00	1.2158	1	1	$34,868,813.30	0	
2121	W	$208,929,432.00	1.7521	1	1	$366,065,257.81	0.0014	
2122	W	$8,888,354.00	1.1982	1	1	$10,650,025.76	0.003775184	
2123	W	$78,946,492.00	1.019264	1	1	$80,467,317.22	0.343687486	
2124	W	$86,045,207.00	0.6811	1	1	$58,605,390.49	0.181011391	
6112	W	$2,153,726.00	1.00767519	1	1	$2,170,256.26	0	
6113	W	$794,298.00	1.007630049	1	0	$0.00	0	
6114	W	$93.00	1	1	1	$93.00	0	
6115	W	$32,698.00	1.007630669	1	1	$32,947.51	0	
6116	W	$602,045.00	0.99284993	1	1	$597,740.34	0	
6121	W	$25,389,861.00	0.916385992	1	1	$23,266,912.96	0.0014	
6122	W	$3,772,263.00	0.850345343	1	1	$3,207,726.27	0.003775184	
6123	W	$4,495,027.00	0.825838691	1	1	$3,712,167.21	0.343687486	

Result set record count: 58

Current Path: C:\ICM\database\

Note: The numbers shown in the above illustration are for presentation purposes only.

COLUMN HEADING	COLUMN DESCRIPTION
ACCOUNT	FCC Part 32 account number
CATEGORY	Indicates Wholesale (W) or Retail (R) information
AMOUNT	Based on regulated expense and investment data as reported in the FCC ARMIS Joint Cost Report (43-03) adjusted for any applicable normalization entries.
ADJUST1	**Expense Adjust1:** Factor used to adjust the ARMIS account balances for such items as avoided cost (wholesale only), billing and collection, directory cost, right-to-use fees, non-recurring activities and any applicable accounting normalization entries. **Investment Adjust1:** The first investment adjustment applies a C.A. Turner index to each FCC Part 32 investment account in order to adjust the booked average plant investment balance to current reproduction cost.
ADJUST2	**Expense Adjust2:** Factor used to adjust ARMIS account balances for expenses identified in the activity-based costing approach (i.e. – service assurance, sales, marketing and advertising, etc.). **Investment Adjust2:** Not Currently used.

COLUMN HEADING	COLUMN DESCRIPTION
ADJUST3	Factor used to adjust the ARMIS expense and investment account balances for the following items: **Expense Adjust3:** 1) Elimination of non-forward looking costs such as account 2211/6211 (Analog Switch), 2215/6215 (Electromechanical Switch), and 2431/6431 (Aerial Wire). 2) Elimination of account balances in which the combination of ADJUST1 and ADJUST2 exceeds 100%. 3) Adjustment to calibrate the investment data associated with Switch, Circuit Equipment, and Outside Plant accounts, to the levels calculated by the Switch, Loop, Interoffice Transport, and SS7 Modules in ICM. **Investment Adjust3:** 1) Eliminate the entire Adjusted ARMIS investment amount for a given FCC Part 32 account for non-forward looking costs, such as account 2211 (Analog Switch), 2215 (Electro-mechanical Switch) and 2431 (Aerial Wire). Adjustment to calibrate the C.A. Turner adjusted ARMIS investment data associated with switch, circuit equipment and outside plant (OSP) accounts to the levels calculated by the Switch, Loop, Interoffice Transport and SS7 modules of ICM.
NEW AMOUNT	Represents the regulated expense and investment data included in the column titled "Amount" adjusted for the effect of Adjust1, Adjust2 and Adjust3. The formula used to calculate the "New Amount" in ICM is as follows: For the computation of adjusted *operating expenses* related to FCC Part 32 accounts 6112 through 6728 and 7240: AMOUNT * [[1 – (1 – Adjust1) – (1 – Adjust2)] * Adjust3] For the computation of adjusted *investment related to general support assets* (i.e.- FCC Part 32 accounts 2111 through 2124): AMOUNT * Adjust1 * Adjust2 * Adjust3

COLUMN HEADING	COLUMN DESCRIPTION
% BC	Percentage of each FCC Part 32 account mapped to the Billing & Collection cost pool associated with Billing and Collection activities.
% CONSUMER	Percentage of each FCC Part 32 account mapped to the Consumer cost pool associated with the Consumer Line of Business activities.
% BUSINESS	Percentage of each FCC Part 32 account mapped to the Business cost pool associated with the Business Line of Business activities
% CXR	Percentage of each FCC Part 32 account mapped to the CXR cost pool associated with the Carrier Line of Business activities.
% COMMON	Percentage of each FCC Part 32 account mapped to the Common cost pool associated with activities that are not directly attributable to the network or a line of business.
$ BC	Portion of the amount of each FCC Part 32 account mapped to the Billing & Collection cost pool associated with Billing and Collection activities. This amount is calculated as follows: $ BC = NEW AMOUNT * % BC
$ CONSUMER	Portion of the amount of each FCC Part 32 account mapped to the Consumer cost pool associated with the Consumer Line of Business activities. This amount is calculated as follows: $ CONSUMER = NEW AMOUNT * % CONSUMER
$ BUSINESS	Portion of the amount of each FCC Part 32 account mapped to the Business cost pool associated with the Business Line of Business activities. This amount is calculated as follows: $ BUSINESS = NEW AMOUNT * % BUSINESS
$ CXR	Portion of the amount of each FCC Part 32 account mapped to the CXR cost pool associated with the Carrier Line of Business activities. This amount is calculated as follows: $ CXR = NEW AMOUNT * % CXR
$ COMMON	Portion of the amount of each FCC Part 32 account mapped to the Common cost pool associated with activities that are not directly attributable to the network or a line of Business. This amount is calculated as follows: $ COMMON = NEW AMOUNT * % COMMON

COLUMN HEADING	COLUMN DESCRIPTION
SUBTOTAL	Represents the amount of general support investment (i.e.- FCC Part 32 accounts 2111 through 2124) assigned to the common cost pools and computed in ICM as follows: $ BC + $ CONSUMER + $ BUSINESS + $ CXR + $ COMMON
ICM BASED FACTORS	Represents the sum of the capital cost factors (i.e.- Depreciation & Return, Composite Income Tax and Property Tax) associated with general support investments utilized in the computation of capital carrying charges. The capital carrying costs for these general support facilities are included in the determination of total common costs. The ICM BASED FACTORS are a sum of: (Depreciation & Return factor + Composite Inc Tax factor + Property Tax factor)

COLUMN HEADING	COLUMN DESCRIPTION
TOTAL	Represents total common costs as computed in ICM as follows: (a) For FCC Part 32 accounts 2111 through 2124: Capital Carrying Costs for general support assets mapped to common cost pools = SUBTOTAL * ICM BASED FACTORS (b) For FCC Part 32 accounts 6112 through 6728 and 7240: Operating Expenses mapped to common cost pools = $ BC + $ CONSUMER + $ BUSINESS + $ CXR + $ COMMON (c) Total Common Costs = Capital Carrying Costs for general support assets mapped to common cost pools + Operating Expenses mapped to common cost pools

APPENDIX M: Output Report – Shared Cost

Purpose: This report contains summarized data, by FCC Part 32 account, of the costs computed in ICM related to:

- Network Services Shared Cost
- Common Cost
- Comparison Price Fixed Allocator

How to Run this Report:

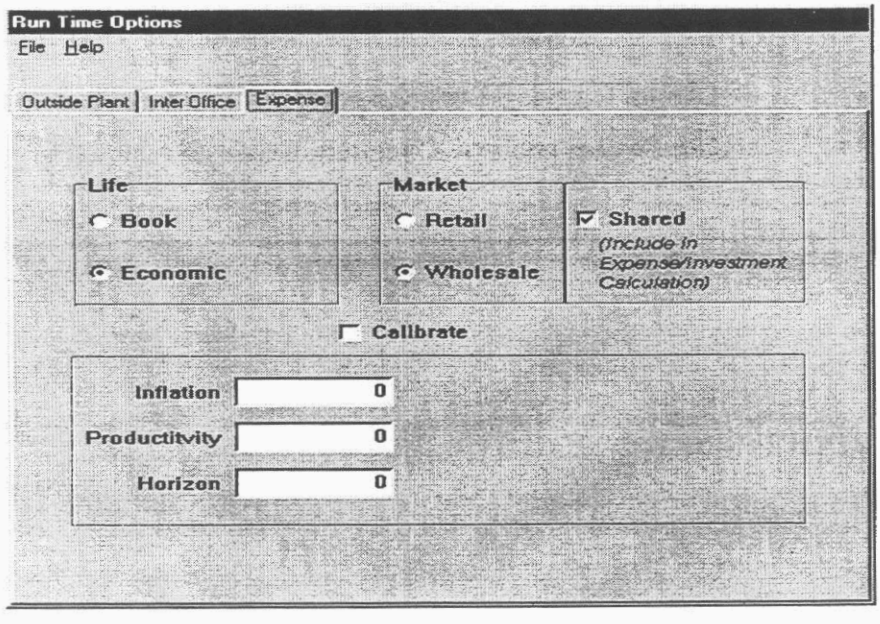
(A) Select Report Parameters:

First, select the parameters necessary to ensure accurate reporting in ICM. Using the drop-done menu select:

Options/Users/Expense tab

Figure M.1 shown below, illustrates the user defined parameter selection screen found in ICM. A complete listing of all data entry fields is found in the section following Figure M.1.

Figure M.1



Note: The selections shown in the above illustration are for presentation purposes only.

Category Heading: **Category Description:**

CATEGORY HEADING	CATEGORY DESCRIPTION
LIFE	Toggle indicates whether "Book" or "Economic" depreciable lives and salvage will be used by ICM in the calculation of Depreciation & Return and Composite Income Tax factors.
MARKET	Toggle indicates whether "Retail" or "Wholesale" expense input data is to be utilized by ICM. ▪ Wholesale data is selected when results are being produced for Unbundled Network Elements (UNE's). ▪ Retail is selected when results are being produced for products and services.
SHARED	Toggle indicates whether "Shared" costs are to be "included" or "excluded" in the calculation of the Maintenance & Support factors within ICM related to network elements. A check mark in this field ensures that shared costs are to be *included* as a component of total expenses used in the computation of Maintenance & Support factors within ICM.
CALIBRATE	Toggle indicates whether ICM should automatically calibrate the C.A. Turner adjusted ARMIS investment data in ICM associated with Switch, Circuit Equipment and Outside Plant activity to the levels calculated by the Switch, Loop, Interoffice Transport and SS7 modules of ICM. A check mark in this field results in the computation of calibration factors with the resulting factors appearing in the Adjust3 column of the XXGTEEXP database file for the following accounts: **Category:** **FCC Part 32 account(s):** Switching 2212 Circuit Equipment 2232 Outside Plant (OSP) 2411 through 2441, excluding 2431
INFLATION	A user-defined inflation factor applied to expense data in ICM over a defined time frame.
PRODUCTIVITY	A user-defined productivity factor applied to expense data in ICM over a defined time frame.
HORIZON	A user-defined time frame (i.e.- number of years) applicable for inflation and productivity adjustments in ICM.

(B) Running the Shared Cost Report in ICM

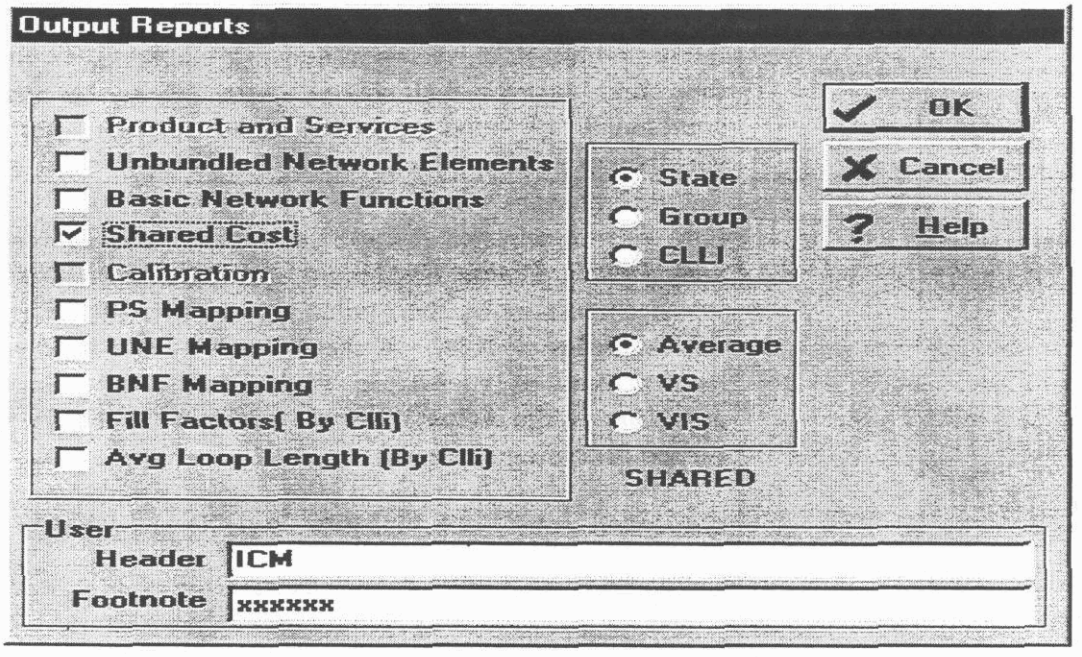
Once you have defined the report parameters it is now time to run the actual report. Using the drop-down menu select:

Report/Output and place a check mark in the "Shared Cost" toggle

Then, using the left button on your mouse, click on the box marked "OK"

Figure M.2 shown below, illustrates the "Output Report" selection screen found in ICM.

Figure M.2



Note: The selections shown in the above illustration are for presentation purposes only.

The Shared Cost calculations generated by ICM produce three separate reports. The three reports include:

- Network Services Shared Cost
- Summary of Common Costs

- Comparison Price Fixed Allocator

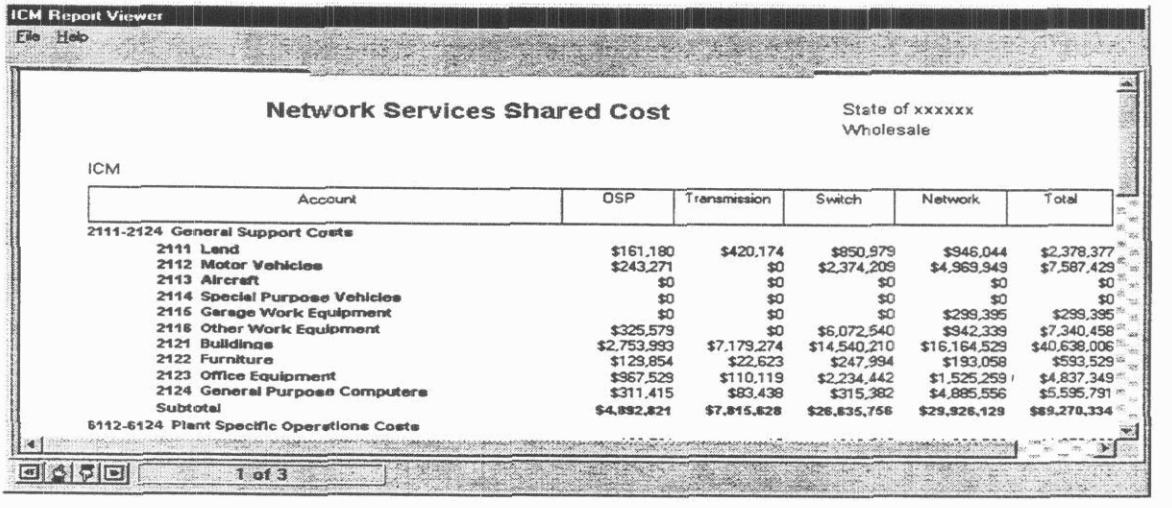
(1) Network Services Shared Cost:

This report details those costs identified as shared expenses associated with Total Service Long Run Incremental Cost (TSLRIC) or Total Element Long Run Incremental Cost (TELRIC).

Shared costs are those costs that cannot be directly associated with a specific element of the network, UNE or service. They are, however, costs that are in direct support of and can be associated with specific groups of services or parts of the network. GTE has identified and developed four categories of shared costs: outside plant, transmission, switching, and network. Within each of these four shared cost categories, the costs have been identified by FCC Part 32 account.

Figure M.3 shown below, illustrates a portion of the data contained within the Network Services Shared Cost report. A complete listing of the information contained within this report is found in the section following Figure M.3.

Figure M.3



Account	OSP	Transmission	Switch	Network	Total
2111-2124 General Support Costs					
2111 Land	$161,180	$420,174	$850,979	$946,044	$2,378,377
2112 Motor Vehicles	$243,271	$0	$2,374,209	$4,969,949	$7,587,429
2113 Aircraft	$0	$0	$0	$0	$0
2114 Special Purpose Vehicles	$0	$0	$0	$0	$0
2115 Garage Work Equipment	$0	$0	$0	$299,395	$299,395
2116 Other Work Equipment	$325,579	$0	$6,072,540	$942,339	$7,340,458
2121 Buildings	$2,753,993	$7,179,274	$14,540,210	$16,164,529	$40,638,006
2122 Furniture	$129,854	$22,623	$247,994	$193,058	$593,529
2123 Office Equipment	$967,529	$110,119	$2,234,442	$1,525,259	$4,837,349
2124 General Purpose Computers	$311,415	$83,438	$315,382	$4,885,556	$5,595,791
Subtotal	$4,892,821	$7,815,628	$26,835,756	$29,926,129	$69,270,334
6112-6124 Plant Specific Operations Costs					

Note: The numbers shown in the above illustration are for presentation purposes only.

COLUMN HEADING	COLUMN DESCRIPTION
ACCOUNT	FCC Part 32 account number
OSP	Outside Plant (OSP) shared cost are those costs that have been determined, through the "Cost Pool" methodology to be related to parts of the network such as poles, aerial, buried and underground cable (copper and fiber) and conduit.
TRANSMISSION	Transmission shared cost are those costs that have been identified to be associated with circuit equipment.
SWITCH	Switching shared cost are those costs that have been associated with Central Office switching equipment.
NETWORK	Network shared cost are those costs related to supervision and management of the total network
TOTAL	Total shared cost computed in ICM. The formula is as follows: OSP + TRANSMISSION + SWITCH + NETWORK

(2) Summary of Common Costs:

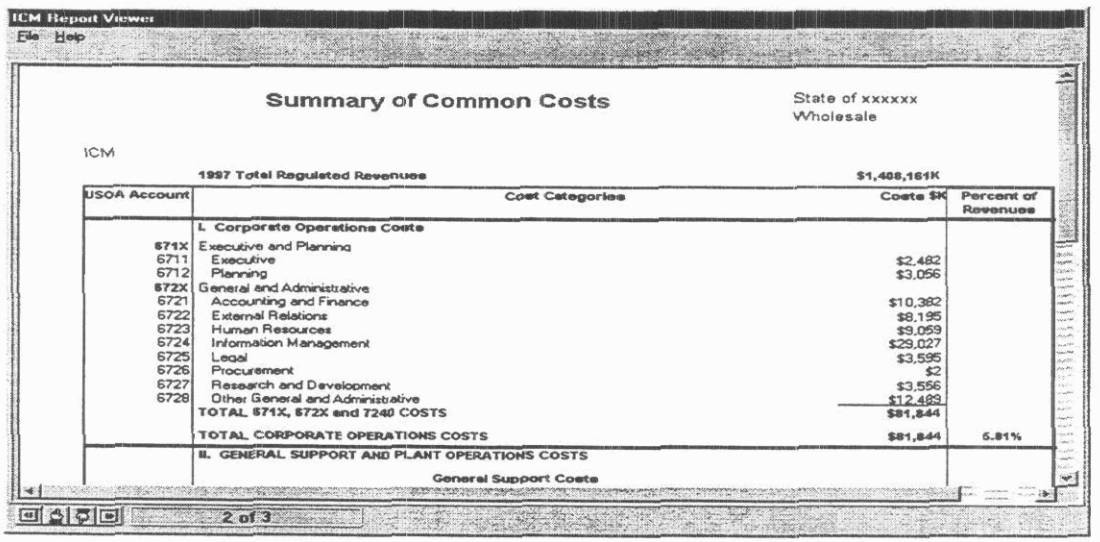
Common Costs relate to all products, services and network elements and can not be directly assigned to any part of the network, subset of the network or the costs of products, services or unbundled network elements.

The computation of common costs within ICM is determined utilizing the "Cost Pool" methodology of identifying operating expenses. Total common costs represents the sum of all FCC Part 32 account balances, after applicable adjustments, assigned to the following cost pools:

- Billing & Collection
- Lines of business (Consumer, Business and Carrier)
- Common

Figure M.4 shown below, illustrates a portion of the data contained within the Common Cost Summary report. A complete listing of the information contained within this report is found in the section following Figure M.4.

Figure M.4



ICM Report Viewer

File Help

Summary of Common Costs

State of xxxxxx
Wholesale

ICM

1997 Total Regulated Revenues **$1,408,161K**

USOA Account	Cost Categories	Costs $K	Percent of Revenues
	I. Corporate Operations Costs		
671X	Executive and Planning		
6711	Executive	$2,482	
6712	Planning	$3,056	
672X	General and Administrative		
6721	Accounting and Finance	$10,382	
6722	External Relations	$8,195	
6723	Human Resources	$9,059	
6724	Information Management	$29,027	
6725	Legal	$3,595	
6726	Procurement	$2	
6727	Research and Development	$3,556	
6728	Other General and Administrative	$12,489	
	TOTAL 671X, 672X and 7240 COSTS	$81,844	
	TOTAL CORPORATE OPERATIONS COSTS	$81,844	5.81%
	II. GENERAL SUPPORT AND PLANT OPERATIONS COSTS		
	General Support Costs		

2 of 3

Note: The numbers shown in the above illustration are for presentation purposes only.

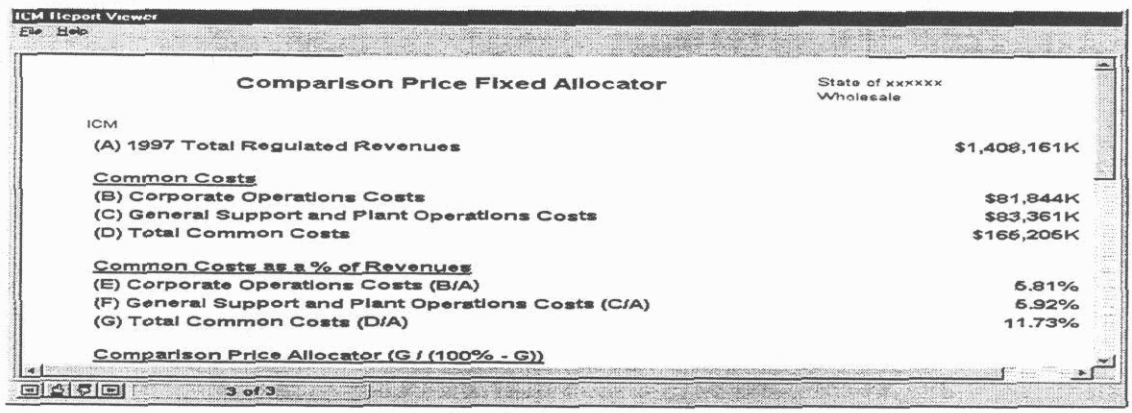
COLUMN HEADING	COLUMN DESCRIPTION
USOA ACCOUNT	FCC Part 32 account number
COST CATEGORIES	Common Costs are detailed by FCC Part 32 account numbers and then summarized into two groups, namely: Corporate Operations Costs and General Support & Plant Operating Costs.
COSTS	Represents regulated ARMIS expense and investment data adjusted for the effect of Adjust1, Adjust2 and Adjust3 that have been mapped specifically to the common cost pools. The formula used to summarize the contents of the individual cost pools is as follows: Billing & Collection + Consumer + Business + CXR + Common
PERCENT OF REVENUES	Represents the computation of common costs as a percent of revenues.

(3) Comparison Price Fixed Allocator:

This report outlines the components and calculation of a Comparison Price Fixed Allocator based on the information contained in the "Summary of Common Costs" report outlined above.

Figure M.5 illustrates a portion of the data contained within the Comparison Price Fixed Allocator report. A complete listing of the information contained within this report is found in the section following Figure M.5.

Figure M.5



Note: The numbers shown in the above illustration are for presentation purposes only.

ROW HEADING	ROW DESCRIPTION
(A) REGULATED REVENUES	Represents total regulated revenues as reported in the annual FCC ARMIS Joint Cost Report (43-03).
(B - D) COMMON COSTS	Represents the Corporate Operations Costs, General Support and Plant Operations Costs and Total Common Costs as computed on the "Summary of Common Costs" report detailed above.
(E - G) COMMON COSTS AS A % OF REVENUES	Represents the percent of revenues computed for Corporate Operations Costs, General Support and Plant Operations Costs and Total Common Costs as computed on the "Summary of Common Costs" report detailed above.
(H) COMPARISON PRICE ALLOCATOR	Comparison price allocator computed in ICM as follows: $\frac{\text{Total Common Costs as a \% of Total Regulated Revenues}}{(\ 100\% - \text{Total Common Costs as a \% of Total Regulated Revenues})}$

This is a Blank Page

This is a Blank Page



Integrated
Cost Model

Release 4.1

Model Methodology

Mapping/Report Module

(Book VII of VII)

GTE Service Corporation

Copyright © GTE 1997 - 2000. All rights reserved.

This is a Blank Page

ICM MODEL METHODOLOGY PACKAGE

- Conceptual Framework — Book I
- Loop Module — Book II
- Switch Module — Book III
- Interoffice Transport Module — Book IV
- SS7 Module — Book V
- Expense Module — Book VI
- Mapping/Report Module — Book VII

This is a Blank Page

Table of Contents

ICM MODEL METHODOLOGY PACKAGE ..... a

MAPPING/REPORT MODULE 1

Output Reports 1

Basic Network Functions (BNF) Report 2

Unbundled Network Element Report 2

Products and Services Report 3

BNF, UNE, and Products and Services Mapping Report 3

Loop Reports 3

APPENDIX A: Data Input Tables 4

Product and Services [PSMAPXX.DB] 6

Unbundled Network Elements (UNE) MAPPING [UNEMAPXX.DB] 7

Basic Network Functions (BNF) MAPPING [BNFMAPXX.DB] 8

Universal Service Fund (USF) [USFMAPXX.DB] 9

Mapping Groups (MGroup) [XXMGROUP.DB] 10

Products and Service Groups (Groups) 11

Wizard Components [CMPNT.DB] 12

Additional Mapping Variables (External) [MAPVAR.DB] 13

Processing Flow 14

This is a Blank Page

MAPPING/REPORT MODULE

The Mapping/Report Module[1] of the Integrated Cost Model (ICM) uses the calculations from the other five Modules (Loop, Switch, Interoffice Transport, SS7 and Expense) and the data from special studies outside the model to calculate the costs of Basic Network Functions (BNFs)[2], Unbundled Network Elements (UNEs), and services. Investment components are developed in the Loop, Switch, Transport and SS7 Module. The Expense Module calculates cost components associated with network investments. The components are mapped to form the monthly costs, per minute and per call costs for BNFs, UNEs, and products and services.

The mapping of components is flexible and can be modified by the user. See the *ICM User Guide* for instructions on how to change the mapping.

Reports can be printed from the model, which show the mapping and monthly costs for BNFs, UNEs, and services. The reports can be previewed, printed or exported into a file; see the *ICM User Guide* for instructions. The output reports are described below.

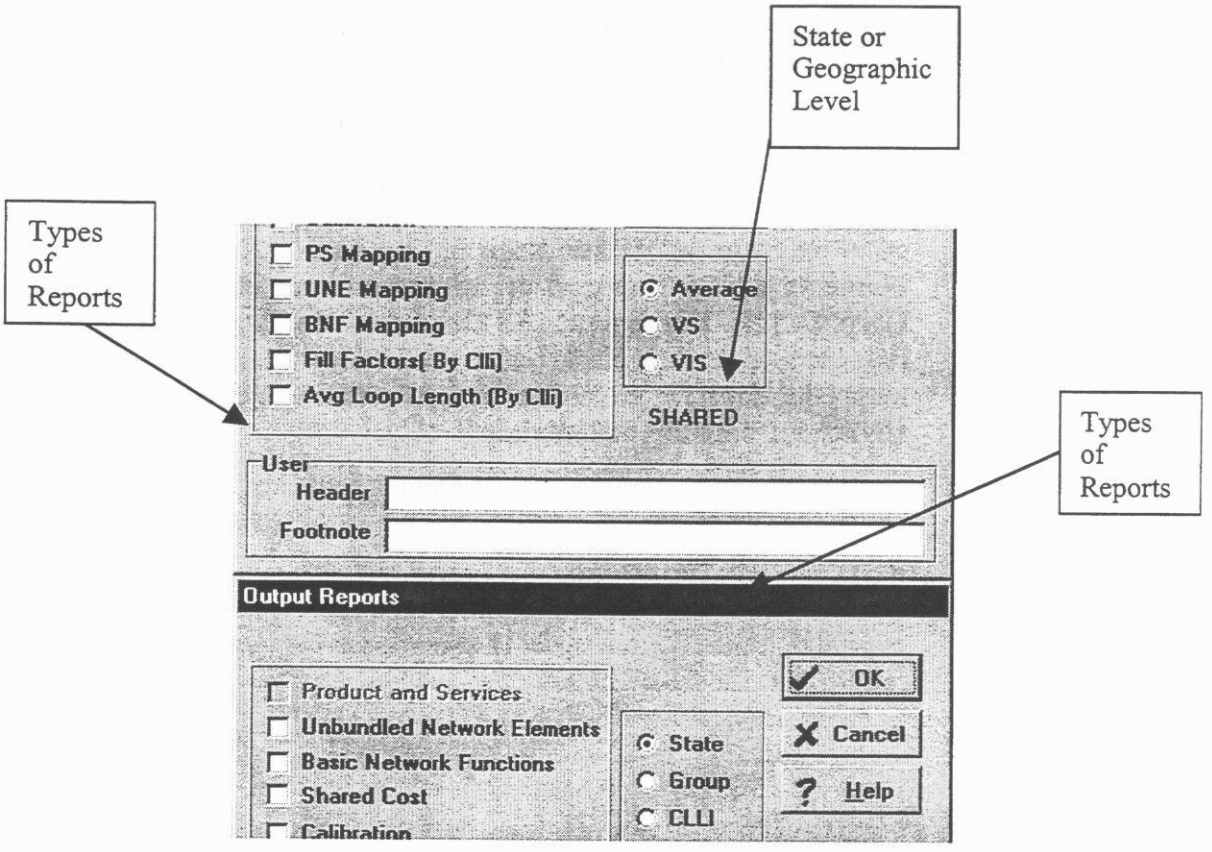
Output Reports

Output reports may be printed by CLLI code, by geographic group or for the entire state. Average costs are produced for BNFs, UNEs, and products and services. Volume sensitive outputs are available for products and services[3]. Each of these reports may be selected at the user's option and is described below. The following picture gives a graphic presentation of Output Reports screen.

[1] The Mapping/Reports Module Methodology is (Book VII of VII) within the ICM Model Methodology package. The contents of the package include seven booklets: Conceptual Framework (Book I); Loop Module (Book II); Switch Module (Book III); Interoffice Transport Module (Book IV); SS7 Module (Book V); Expense Module (Book VI); and Mapping/Reports Module (Book VII).

[2] Basic Network Function reports also includes basic network function details.

[3] Volume Sensitive and Volume Insensitive reports are not applicable for BNFs and UNEs.



Basic Network Functions (BNF) Report

The BNF Report provides the total monthly costs for each of the underlying network functions (BNF column). In addition, total investments and costs by category are provided. Cost detail is shown for the following categories:

Depreciation & Return

Composite Income Tax

Property Tax

Maintenance & Support

Unbundled Network Element Report

The UNE Report provides the Total Element Long Run Incremental Cost (TELRICs). Monthly costs are provided for non-usage sensitive elements and costs per unit are provided for usage sensitive. In addition, total investments and costs by category are provided. Cost detail is shown for the following categories:

- Depreciation & Return
- Composite Income Tax
- Property Tax
- Maintenance & Support
- Marketing
- Billing/Collection & Directory

Products and Services Report[4]

Products and Services costs are reported as Total Service Long Run Incremental Cost (TSLRICs) on an average or volume sensitive basis. Investment for each product and service is provided, and the annual cost for the following cost categories are shown:

- Depreciation and Return
- Composite Income Tax
- Property Tax
- Maintenance and Support (includes cost pool and service-specific activity based costs)
- Marketing (includes service-specific activity based costs)
- Billing/Collection and Directory (from external studies)

NOTE: The TSLRICs provided on this report do not include toll the usage cost components. The toll usage cost components is mapped to products and services in ICM-SAS. TSLRIC for toll services are provided by ICM-SAS.

BNF, UNE, and Products and Services Mapping Report

The Mapping Report provides information regarding how the network components are combined to develop costs for the UNEs. The mapping can be modified by the user to develop custom reports. To print a custom report, the modified mapping table needs to be added to the Associate table.

Loop Reports

Two reports that provide loop information are also provided by ICM. For each CLLI code, the Fill Factor Report provides the cable utilization calculated by ICM for both feeder and distribution; the Average Loop Length Report provides the average loop length.

[4] Not applicable for states for which products and services are not provided.

APPENDIX A: Data Input Tables

The data input tables are listed below. The tables can be viewed and printed under the menu option **Tables/ Product Mapping.** The name of each database file is listed; often the first two letters indicate state (that is, in xxexbase.db, 'xx' indicate a state's abbreviation). A user may modify data in all of the tables listed below. Detailed descriptions of each of the tables are provided in the following section.

Following the description of data input tables, a "bubble chart" for Mapping/Reports is provided under the heading 'Processing Flow'. The charts contain a graphic presentation of the database files that are inputs to and outputs of the Mapping/Report Module.

Product Mapping

- Product and Services
  - List (PSMAPXX.DB)
  - Mapping (PSMAPXX.DB)
- Unbundled Network Elements
  - List (UNEMAPXX.DB)
  - Mapping (UNEMAPXX.DB)
- Basic Network Functions
  - List (BNFMAPXX.DB)
  - Mapping (BNFMAPXX.DB)
- Universal Service Fund
  - List (USFMAPXX.DB)
  - Mapping (USFMAPXX.DB)
  - Header Info (USFMAPXX.DB)

- Mapping Group
  - Mgroups (XXGROUP.DB)
- Product and Service Group
  - Groups
- Wizard Components
  - Components (CMPNT.DB)
- External
  - Table Definitions (MAPVAR.DB)
  - Map Variables (MAPVAR.DB)
- Processing Flow
  - Mapping (Step One, Two and Three)
  - Report

Product and Services [PSMAPXX.DB]

Purpose: Allows the user to design output reports for products and services

How to View On-line: Table/Product Mapping/Product and Services/List Tab

For More Information: See Mapping/Reporting Module.

Note: The Mapping tab shows the cost components that make up each product and service. The user can modify the mapping; see the *ICM User Guide*.

COLUMN HEADINGS	DESCRIPTION
SEQUENCE	Indicates the line number
INDENT	Indicates the number of spaces the line is indented
LABEL	Indicates whether the line contains a heading (true) or a product or service that results will be provided for (false)
FORMAT	Code used to control the format of numbers. (For example, %10.2F creates 10 columns with 2 decimal places and a floating point decimal.)
DESCRIPTION	Indicates the product or services for which results are provided
PS TABLE	Indicates the table names where results are stored
MAPCODE	Algorithm applied to components for computation
EXECODE	Machine Code for the mapcode

Unbundled Network Elements (UNE) MAPPING [UNEMAPXX.DB]

Purpose: Allows the user to develop output reports for unbundled network elements

How to View On-line: Table/Product Mapping/Unbundled Network Elements/List Tab

For More Information: See Mapping/Report Module.

Note: The Mapping tab shows the cost components that make up each unbundled network element. The user can modify mapping; see the *ICM User Guide*.

COLUMN HEADINGS	DESCRIPTION
SEQUENCE	Indicates the line number
INDENT	Indicates the number of spaces the line is indented
LABEL	Indicates whether the line contains a heading (true) or a product or service that results will be provided for (false)
FORMAT	Code used to control the format of numbers. (For example, %10.2F creates 10 columns with 2 decimal places and a floating point decimal.)
DESCRIPTION	Element description defined by the user
MAPCODE	Algorithm applied to components for computation
EXECODE	Machine Code for the mapcode
RESULTS	Unbundled Network Elements (UNE) Cost outputs

Basic Network Functions (BNF) MAPPING [BNFMAPXX.DB]

Purpose: Allows the user to develop output reports for basic network functions

How to View On-line: Table/Product Mapping/Basic Network Functions/List

For More Information: See Mapping/Report Module.

Note: The Mapping tab shows the cost components that make up each basic network function. The user can modify the mapping; see the *ICM User Guide*.

COLUMN HEADINGS	DESCRIPTION
ORDER	Indicates the line number
BNF CODE	Code which identifies the basic network function-can be varied by the user
INDENT	Indicates the number of spaces the line is indented
LABEL	Indicates whether the line contains a heading (true) or a product or service that results will be provided for (false)
FORMAT	Code used to control the format of numbers. (For example, %10.2F creates 10 columns with 2 decimal places and a floating point decimal.)
DESCRIPTION	Indicates the product or services for which results are provided
MAPCODE	Algorithm applied to components for computation
EXECODE	Machine Code for the mapcode

Universal Service Fund (USF) [USFMAPXX.DB]

Purpose: Allows ICM to calculate USF subsidy for select high cost areas

How to View On-line: Table/Product Mapping/Universal Service Fund/List

For More Information: See Mapping/Report Module.

Note: The Mapping tab shows the cost components that make up each Universal Service Fund. The user can modify the mapping; see the *ICM User Guide*.

COLUMN HEADINGS	DESCRIPTION
SEQUENCE	Indicates the line number
INDENT	Indicates the number of spaces the line is indented
LABEL	Indicates whether the line contains a heading (true) or a product or service that results will be provided for (false)
FORMAT	Code used to control the format of numbers. (For example, %10.2F creates 10 columns with 2 decimal places and a floating point decimal.)
DESCRIPTION	Indicates the product or services for which results are provided
MAPCODE	Algorithm applied to components for computation
EXECODE	Machine Code for the mapcode

Mapping Groups (MGroup) [XXMGROUP.DB]

Purpose: Allows user to develop reports for groups of wire centers

How to View On-line: Table/Product Mapping/Mapping Groups

Note: See the *ICM User Guide* for more information regarding how to map groups of CLLIs.

COLUMN HEADINGS	DESCRIPTION
CLLI	Indicates CLLI code of each wire center
PARENT CLLI	Indicates host CLLI code (same as CLLI unless CLLI is a remote)
GROUP	Indicates group that CLLI belongs to and can be varied by the user

Products and Service Groups (Groups)

Purpose: Allows the user to group specific CLLIs for mapping and reporting purposes.

How to View On-line: Table/Product Mapping/Products and Service Groups

Note: See *ICM User Guide* for more information on how to use this function.

COLUMN HEADINGS	DESCRIPTION
MASTER CLLI	Includes all CLLIs for a particular state
GROUP CLLI	List of CLLIs for a particular state from which CLLIs can be selected for extended area service
PROD. AND SERV. CLLI	Selected CLLIs for group for the highlighted Master CLLI

Wizard Components [CMPNT.DB]

Purpose: Defines the valid component names that can be used in mapping. Provides component information to enable the mapping wizard function to operate.

How to View On-line: Table/Product Mapping/Wizard Components

How to Print: View table on-line, then File/Print

COLUMN HEADINGS	DESCRIPTION
COMPONENT	The alphanumeric code used to identify an input defined by the user.
CATEGORY	A letter used to identify to module that will include the user defined component. 7 SS7 A Activity based cost E Expense F Feature L Loop R Rate period S Switching T Transport U USF X Misc.
ACCOUNT	Account Number - See Expense Table for USOA Account Descriptions.
DESCRIPTION	Definition of a component

Additional Mapping Variables (External) [MAPVAR.DB]

Purpose: Allows user to define mapping variable table to import additional CILLI level parameters to mapping – such as usage parameters. And, in addition, allows user to conduct a sensitivity study with, multiple, or more than one column of values

How to View On-line: Table/Product Mapping/External

How to Print: View table on-line, then File/Print

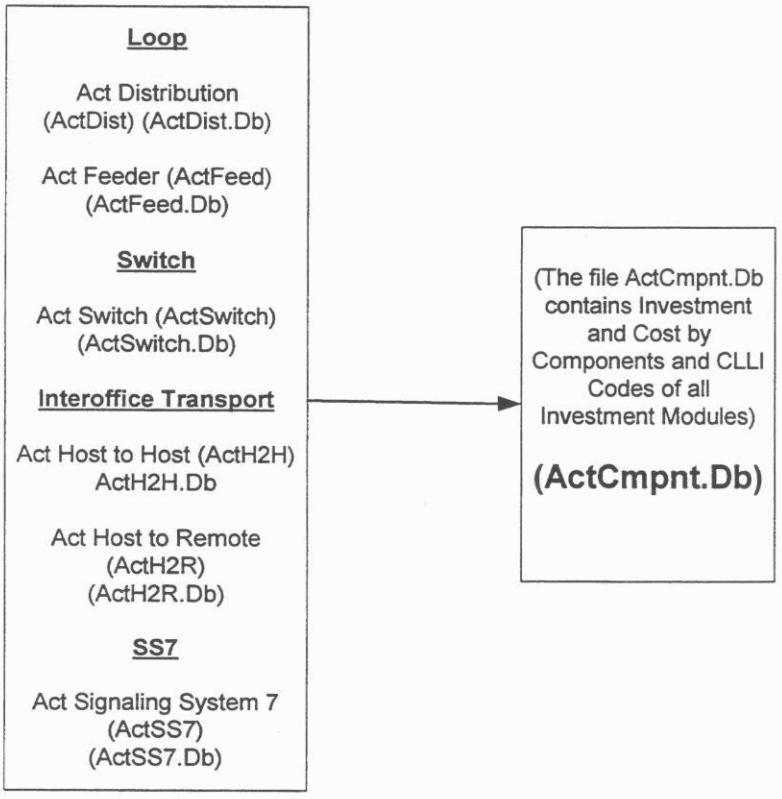
COLUMN HEADINGS	DESCRIPTION
FIELD NUMBER	Indicates the Identification or Roll number
FIELD NAME	Internal Reference field
MAPPING VARIABLE	Alias for column from the component table
DESCRIPTION	Definition of a component

Processing Flow

The Mapping bubble chart provides a graphic view of the database files that are inputs to and outputs of the Mapping portion of the Mapping/Report Module. The mapping processing flow is divided in three steps.

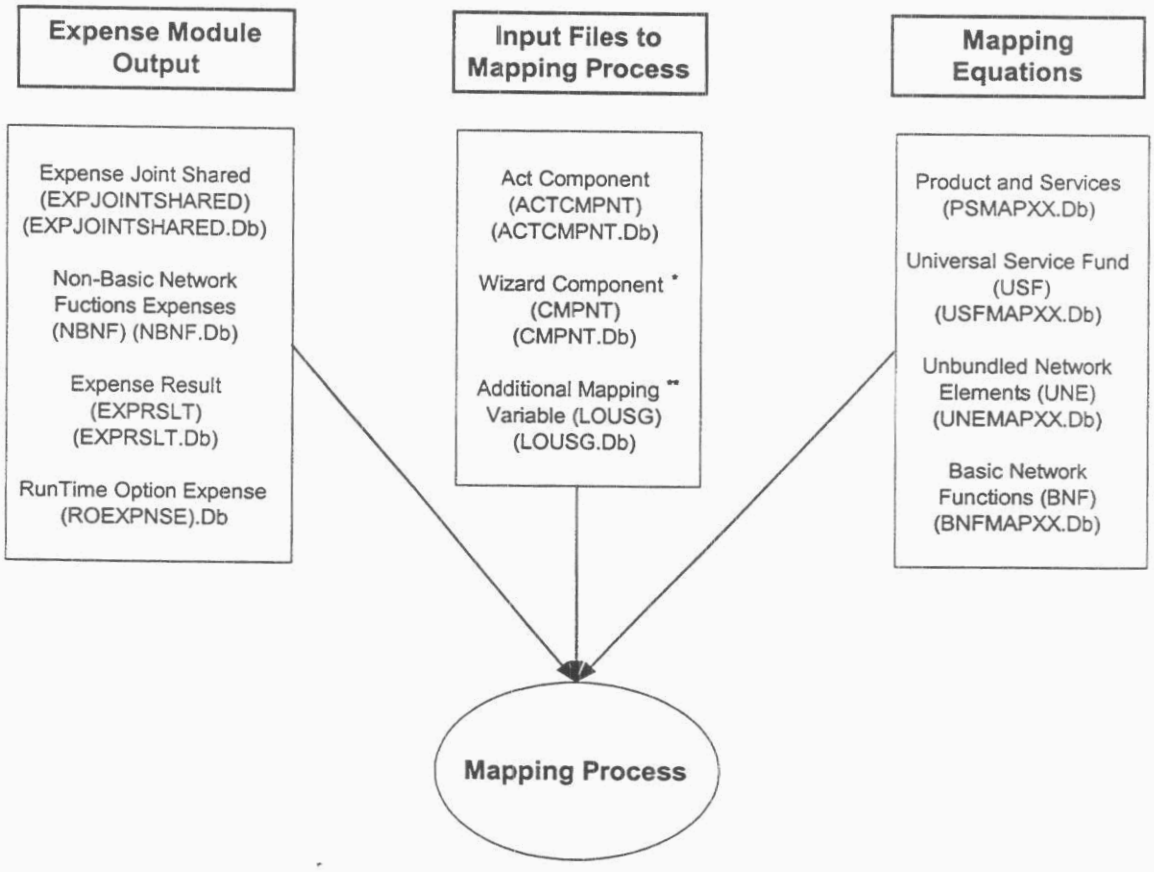
Mapping Step One

Collection of Loop, Switch, Interoffice Transport and SS7 Module's Output(s) in ActCmpnt.Db



Mapping Step Two

Inputs to Mapping Process : ActCmpnt.Db, Wizard Component, Additional Mapping Variable, Expense Module Output and Mapping Equations

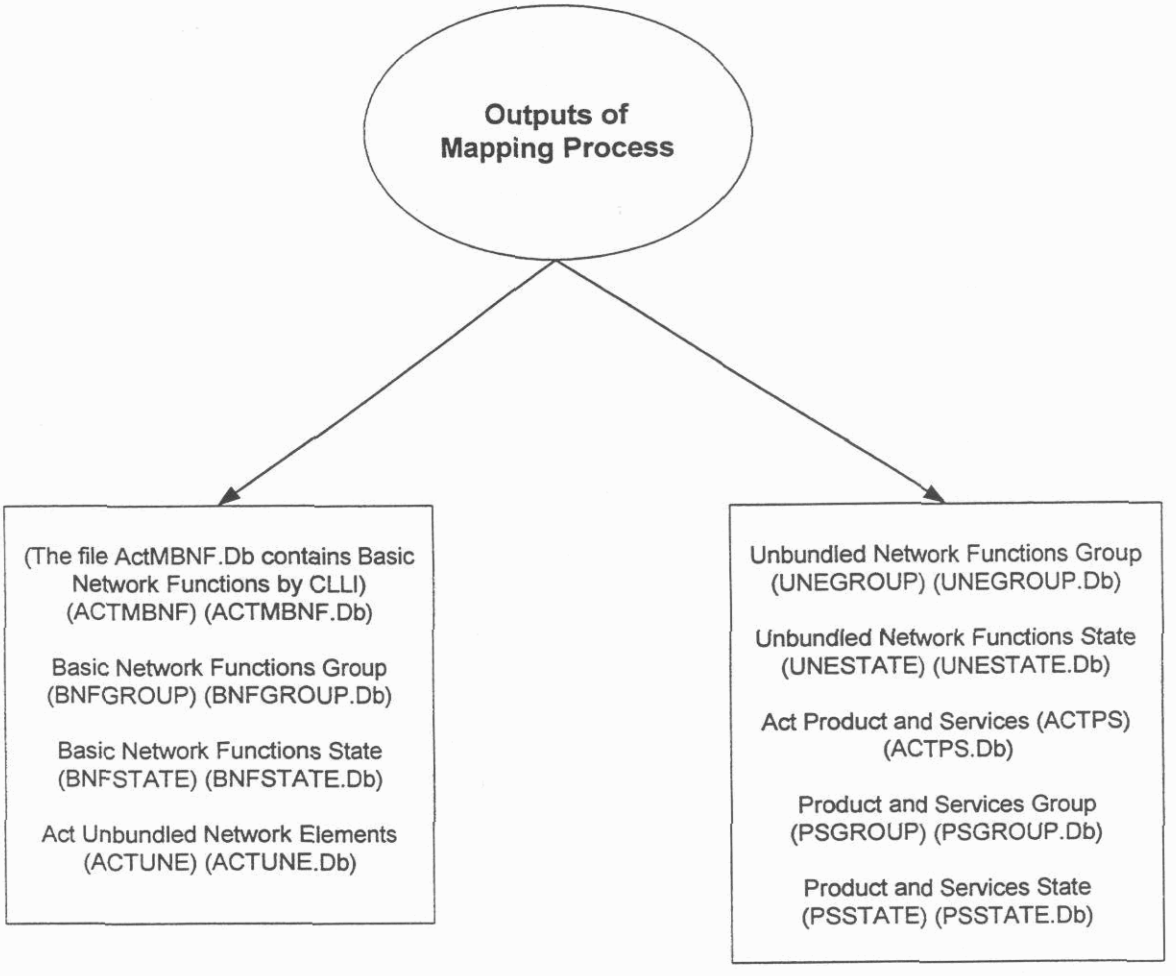


* Wizard Component: Enables user to automatically map expense and investment components from Expense and Investment Modules.

** Additional Mapping Variables: User-defined variables. Enables a user to input state-specific variables at CLLI level.

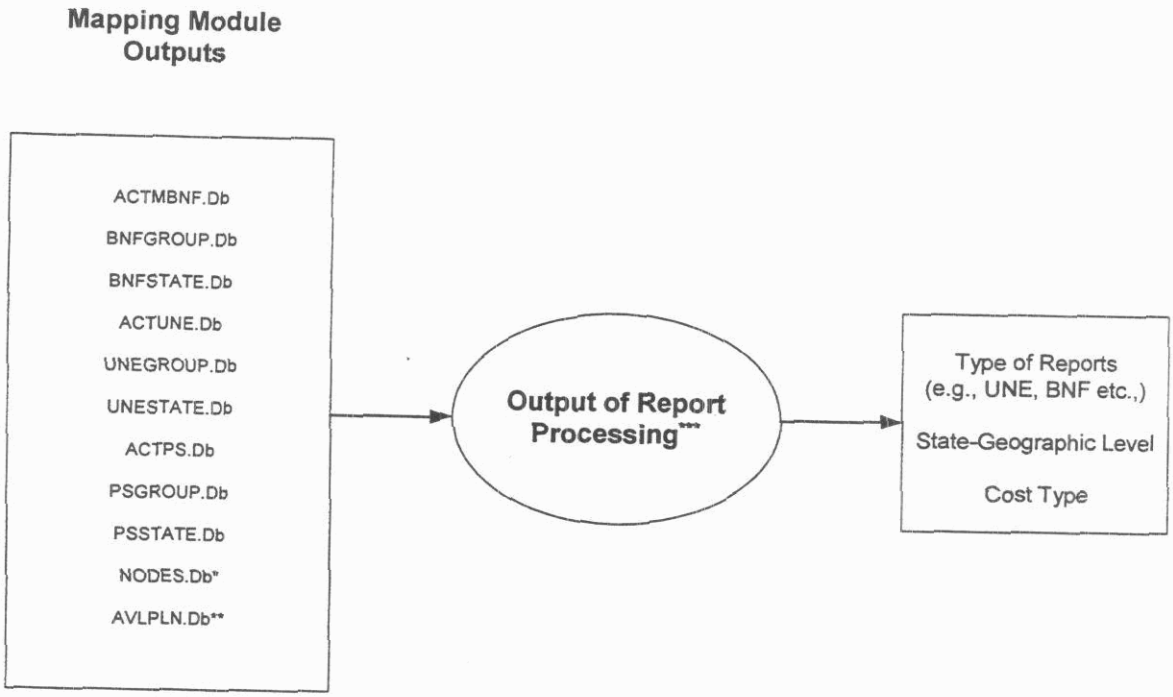
Mapping Step Three

Outputs of Mapping Process



The Report bubble chart provides a graphic view of the database files that are inputs to and outputs of the Report portion of the Mapping/Report Module.

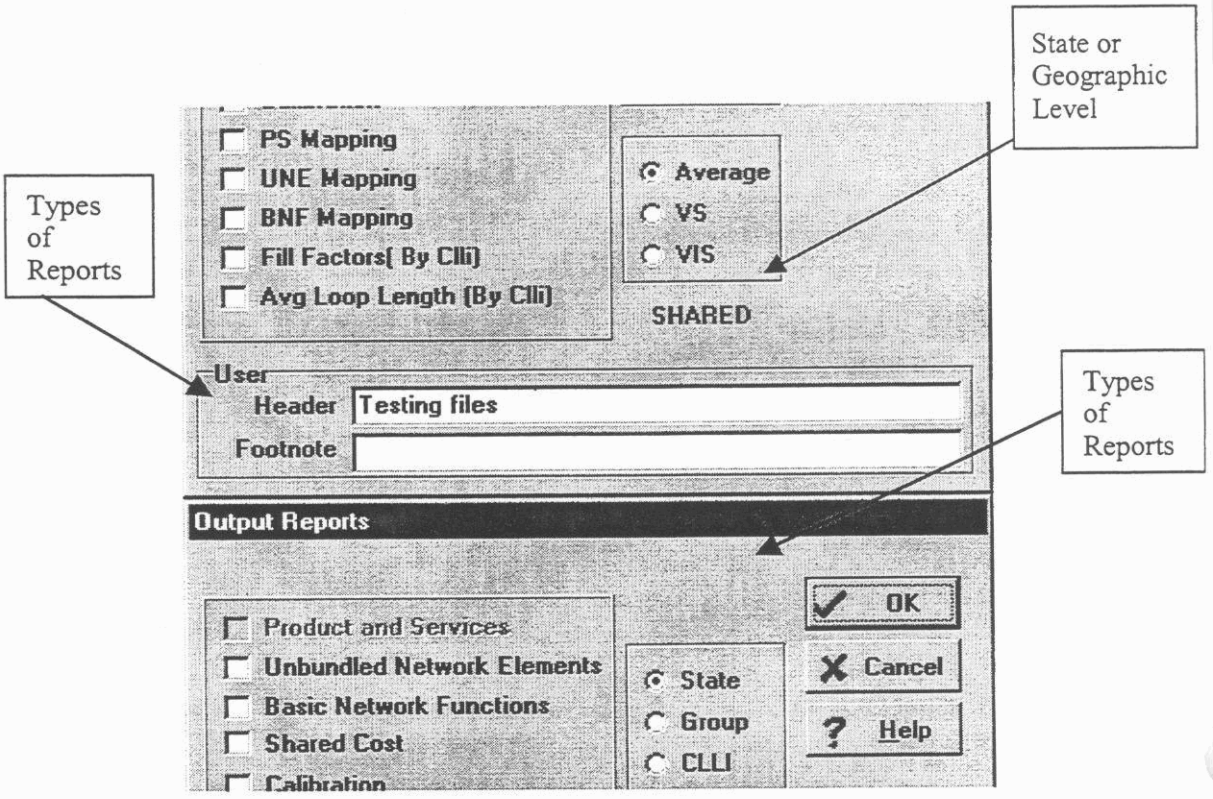
Report



* Nodes.Db : Input file from the processing of all investment modules.

** Average Loop Plant (AVLPLN.Db) : Intermediate file from the processing of the loop module.

*** See below for a graphic view of the Output Report Screen.



The diagram above shows three sections from which a user may select specific items to tailor a report: Type of Report section, State or Geographic Level section and Cost Type section.

This is a Blank Page

This is a Blank Page



Integrated
Cost Model

Release 4.1

User Guide

GTE Service Corporation

Copyright © GTE 1997 - 2000. All rights reserved.

Integrated Cost Model User Guide

Copyright © GTE 1997 - 2000

All rights reserved.

Pentium is a trademark of Intel Corporation.

Table of Contents

Introduction to ICM 1

ICM Overview 1
About this Guide 1
Conventions 1
Using Online Help 2
Technical Support 2

Getting Started 3

System Requirements 3
Installing ICM 4
Starting ICM 5
Running Multiple Versions of ICM 5
Setting the Monitor Level 6

Generating Reports 7

Generating Input and Output Reports 7

Viewing Data and Code 9

Viewing Input Data 9
Viewing Loop Module Data 9
Viewing Switch Module Data 15
Viewing Interoffice Module Data 19
Viewing Signaling System 7 (SS7) Module Data 24
Viewing Expense Module Data 25
Viewing and Searching Data Tables 28
Viewing Table Data 29
Searching for Table Information 30
Entering an SQL Statement 32
Saving Search Results 32
Selecting How the Columns Sort 32
Summing a Single Column 33
Finding the Highest Value in a Column 33
Viewing Demand Information, Cluster and Feeder Route to Digital Loop Carrier (DLC). 35
Viewing Source Code 39

Editing Input Data and Mappings 40

Editing Data 40
Editing Data Tables 40
Importing Data Files 43
Building or Rebuilding the Interoffice Tables 45
Managing Multiple Sets of Data Files 46
Exporting Specific Input Data Tables 47
Editing Run Time Options 49
Editing the Mappings 49
Editing the Product Mapping 49
Editing the Mapping Groups 52
Setting Up Product and Service Groups 54
Running the Model 55

Recreating Index Files 5

Appendix A: Programming Language Alpha (PL-A) 5

PL-A General Considerations and Requirements 5
Operators 5
Assignment of values to variables 5
Flow Control Statements 5
Math functions 5
Mapping Examples 6

Appendix B: Using SAS to Develop Final Service Costs 6

Installing SAS 6
Exporting the BNF Output Report to SAS 6

Introduction to ICM

ICM Overview

The Integrated Cost Model (ICM) has been designed to streamline the cost study process within GTE and make GTE cost studies more open and user-friendly for outside parties. The model features an easy-to-use, Windows-based user interface. Cost study inputs and outputs are presented in a table format that can be viewed or printed. To facilitate sensitivity analyses, the user can easily change inputs. This guide describes how to install ICM, run reports, view data, etc.

For detailed information about the methodology used to develop the Integrated Cost Model, refer to the various modules in the *ICM Model Methodology package*[1].

About this Guide

Conventions

Convention	Definition
Bold	Buttons, menu items, keys you press.
Bold/Bold	Menu selections with the menu title before the slash and the item you select after the slash.
`Courier New font`	Exact names of files and directories.

[1] ICM Model Methodology package contains seven booklets: Conceptual Framework (Book I); Loop Module (Book II); Switch Module (Book III); Interoffice Transport Module (Book IV); SS7 Module (Book V); Expense Module (Book VI) and Mapping/Report Module (Book VII).

Using Online Help

This entire manual is available online. Click **Help** from the menu bar to view the online help.

Technical Support

For technical support, contact the GTE external affairs representative assigned to your state.

Getting Started

System Requirements

Your computer must meet the following minimum requirements to run ICM:

- IBM-compatible PC
- Pentium™ 120 MHz microprocessor
- 16 MB RAM
- 1 GB hard drive space
- Windows 95 or NT
- CD drive

Installing ICM

To install ICM:

1. Insert the ICM CD into your CD drive.
   - The setup program may start automatically. If it does not, continue with step 2.
2. Select **Start/Run**.
3. Click **Browse**.
4. Browse the directories on the ICM CD to find the file called: `setup.exe`.
5. Select the file.
6. Press **Enter**.
   - Follow the instructions displayed on your screen.
   - The system displays a message when the installation is complete.

If you are installing an additional or updated version of ICM, the installation may cause an Error 3 or Alias Error 3. You can avoid this error in one of three ways:

- You just click past the error and then edit the alias. The alias is the path where ICM looks for the Database directory that contains the data files for a specific state. For more information, see "Running Multiple Versions of ICM" on page 5.
- Uninstall the old version of ICM and install the new one in the same directory. The error will still occur, but will have no effect.
- Delete the IDAPI directory before installing. The drawback to this method is that if you have other applications using that API they will be affected.

Starting ICM

To start ICM:

1. Select **Start/Programs/Integrated Cost Model**.
   - The system opens the main window. All instructions in this guide start from this window.

Running Multiple Versions of ICM

If you need to run ICM for more than one state, you can have multiple versions of ICM on your computer. You must install each version in a different directory. You can select the directory during the installation process. You can call the directory what ever you want to help you identify the state.

If you want to install a new version of ICM for the same state, you must first uninstall the old version for that state and then install the new version.

You must tell the system which version you want to run by setting up the alias path. Once you set up the alias path for a state, you can then run a different state by selecting the state as described below.

To set the version you want to run:

1. Select **File/Change Alias Path** or **Options/Set State**.
   - The Select/Define State window displays.
2. Select the state you want to run.
3. Enter the full path to the Database directory of the version of ICM you want to run in the "Define alias path" field.
   - You can click the folder button next to the field to navigate to the correct directory.
4. Click **OK**.
   - The system saves the state and path combination, restarts ICM using that combination, and closes the original version of

ICM. Since two versions of ICM are briefly open at the same time, you may get an insufficient resources error. You can ignore the error.

Setting the Monitor Level

You can set the level of activity detail that ICM monitors. The level of detail can help determine the cause of any errors that arise when using ICM.

Setting a monitor level slows processing. We recommend that most users use the Suspend setting to ensure that processing does not slow down.

To set the monitor level:

1. Select **Utilities/Monitor/Run Level**.
   - The Select Monitor Run Level window displays.
2. Select the level of detail you want.
3. Click **OK**.

The system saves all activities at the level you specify in a log file that you can view.

To view the log file:

1. Select **Utilities/Monitor/View Log**.
   - The Monitor window displays.
   - You can now use the query functions to search for or sort the information. For more information, see "Viewing and Searching Data Tables" on page 28.
2. Select **File/Exit** to exit the window.

Generating Reports

Generating Input and Output Reports

You can generate reports of the ICM inputs and outputs. If you have changed any data in the model, the system recalculates the outputs before generating the report.

For information about viewing input data, see "Viewing Input Data on Page 9. For information on input reports, refer to Reports section in the Preface/ICM Overview Book I within the *ICM Model Methodology package*[2], and Appendix A in the Book I for information on User Variable Inputs. You may also refer to the ICM online help "Input Data".

For information about the output reports refer to the Output Reports section in the Mapping/Report Module (Book VII) within the *ICM Model Methodology* package or the ICM online help "Output Reports."

Generating input or output reports works the same way. The steps below use input reports as the example.

To generate reports:

1. Select **Report/Inputs**.
   - The system displays a list of the available reports.
2. Select the reports that you want.
   - You can select more than one report.

[2] ICM Model Methodology package contains seven booklets: Conceptual Framework (Book I); Loop Module (Book II); Switch Module (Book III); Interoffice Transport Module (Book IV); SS7 Module (Book V); Expense Module (Book VI) and Mapping/Report Module (Book VII).

- For output reports only, you can enter text that you want to print in the header or footnote.

3. Click **OK**.
   - The system sends each input report you selected to your default printer.
   - Output reports display after they are processed. You can then print the report or select **File/Save As** to save the report as a text file.
   - Output reports let you select from two sets of options: State, Group, or CLLI (Common Language Location Identifier) and Average, VS, and VIS:
     - The State, Group, or CLLI options let you select whether you want the report to include the whole state, a group, or a specific CLLI.
     - The Average option creates a report using the average investment. The VS option creates a report using the volume-sensitive investment, and the VIS option uses the volume-insensitive investment.
   - The Basic Network Functions output report is used in external studies to develop service costs. You can export the report to SAS to develop final usage-related service costs. For more information, see "Appendix B: Using SAS to Develop Final Service Costs" on page 64.
   - The Product and Services output report is available only when you select **Options/User**, click the Expense tab, and select Retail.
   - The Unbundled Network Elements report is available only when you select **Options/User**, click the Expense tab, and select Wholesale.

Viewing Data and Code

Viewing Input Data

You can view the input data for each ICM Module. The primary data for each Module is available from the Tables menu. User options are available from the Options menu. For information about editing input data, see "Editing Input Data and Mappings" on page 40.

Viewing Loop Module Data

You can view demand data, the outside plant material and placement costs, and other outside plant inputs.

For information about editing input data, see "Editing Input Data and Mappings" on page 40. For information about the Loop Module data see Appendix A in the Loop Module Book II within the *ICM Model Methodology package*, or the ICM online help "Input Tables."

A demand cell is the geographic building block that ICM uses. It represents an area of approximately 1,500' by 1,800'.

Viewing Demand Data

To view demand data:

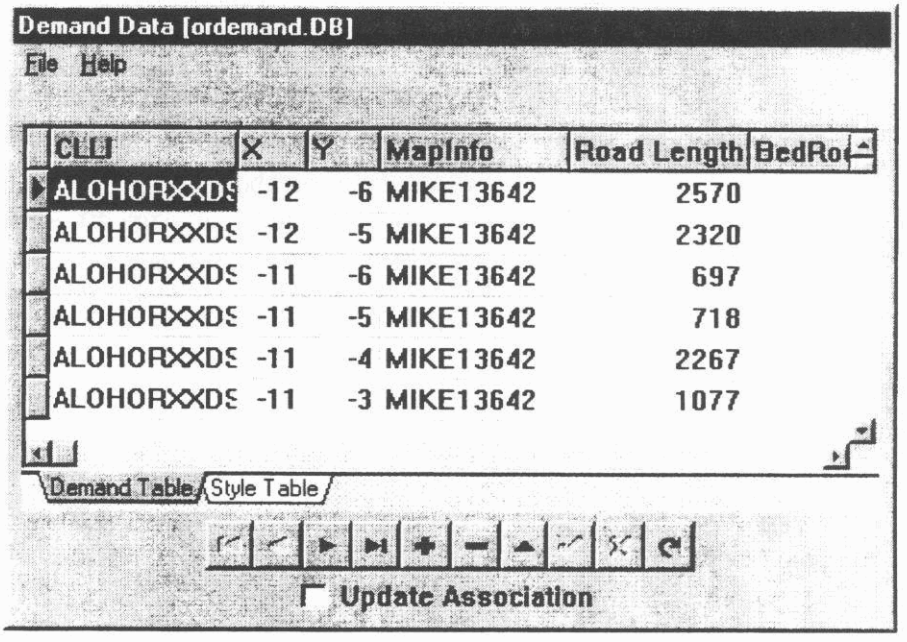
1. Select **Table/Loops/Demand**.

   - The system displays the Demand Data window which has two tabs: Demand Table tab and Style Table tab.

     Demand Table tab lists each demand cell characteristics - CLLI, horizontal and vertical location, road length etc. And Style Table tab specifies the nine distribution styles by number of road feet. .

   - To print the contents of the current tab, select **File/Print**.

2. Select **File/Exit** to return to the main window.



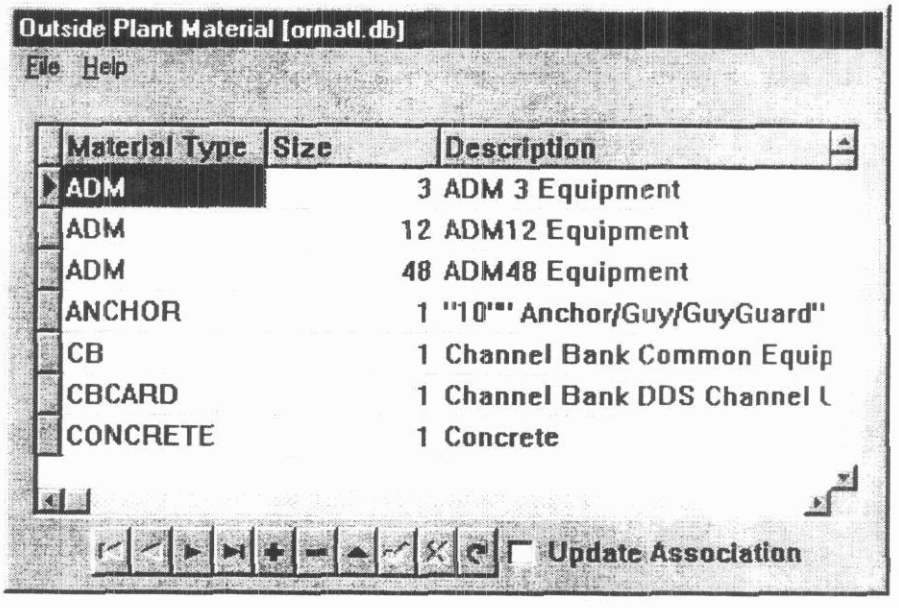
Note: The screen shot is for illustrative purposes only.

Viewing Outside Plant Material or Placement Costs (Loop)

- For information about editing input data, see "Editing Data" on page 40. For information about the Loop Module data, see Appendix A Loop Module Book II in the *ICM Model Methodology package,* or the ICM online help "Input Tables."

To view outside plant material costs or placement costs:

1. Select **Table/Loops/Material** or **Table/Loops/Placement Costs**.
   - Below is the Outside Plant Material table.



Note: The screen shot is for illustrative purposes only.

   - To print the contents of the current table, select **File/Print**.
2. Select **File/Exit** to return to the main window.

Viewing Outside Plant User Options

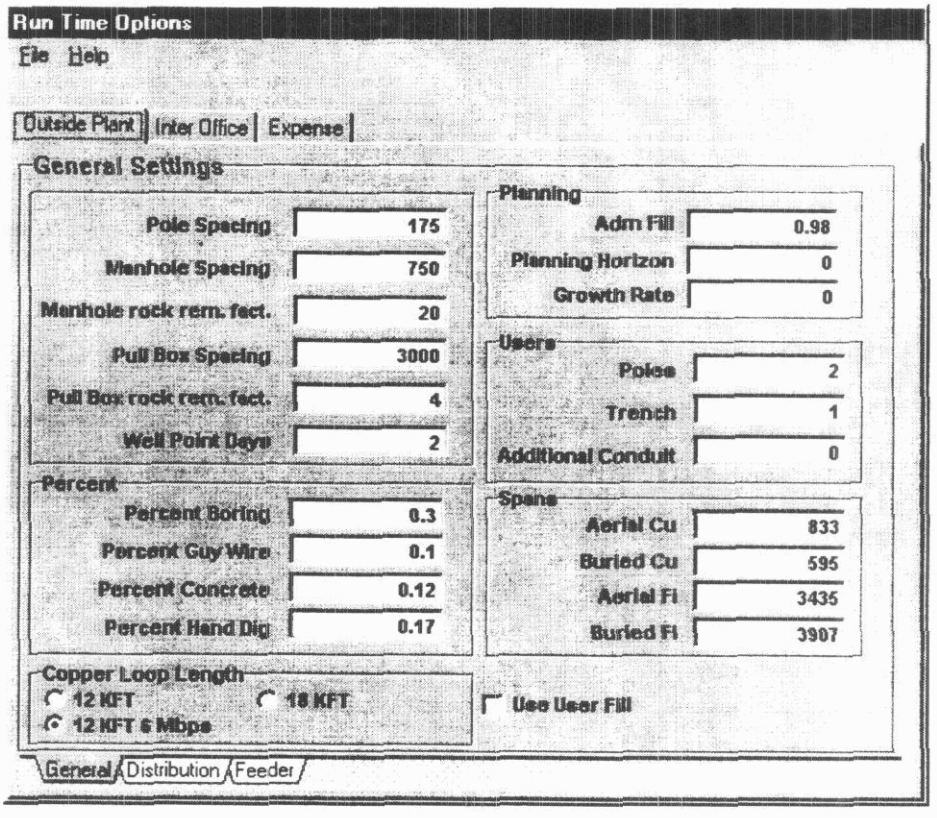
- For information about editing input data, see "Editing Input Data and Mappings" on page 40.

For information about Outside Plant User Option data, see Appendix A in the Conceptual Framework (Book I) within the *ICM Model Methodology package* or the ICM online help "Input Tables."

To view the outside plant user options:

1. Select **Options/User**.

   - The system displays the Run Time Options window that has three tabs. It displays the Outside Plant tab, the Inter Office tab and Expense tab. The Outside Plant tab is further divided in three sub-tabs, General, Distribution and Feeder. Each of the three sub-tabs contain following information.



Note: The screen shot is for illustrative purposes only.

   - The General tab contains structure spacing, and placement assumptions for outside plant facilities.
   - The Distribution tab contains information such Drop Size, Maximum and Minimum Drop Wire length, Business and Residential Threshold etc.

- The Feeder tab contains information such as Feeder line and unit weight, Feeder sharing, cross connect box etc.

- The Interoffice tab contains the assumptions to calculate facilities sharing, placement, and cable sizing.
- The Expense tab contains depreciation life and market options (retail or wholesale).
- To print the User Options, select **File/Print**.

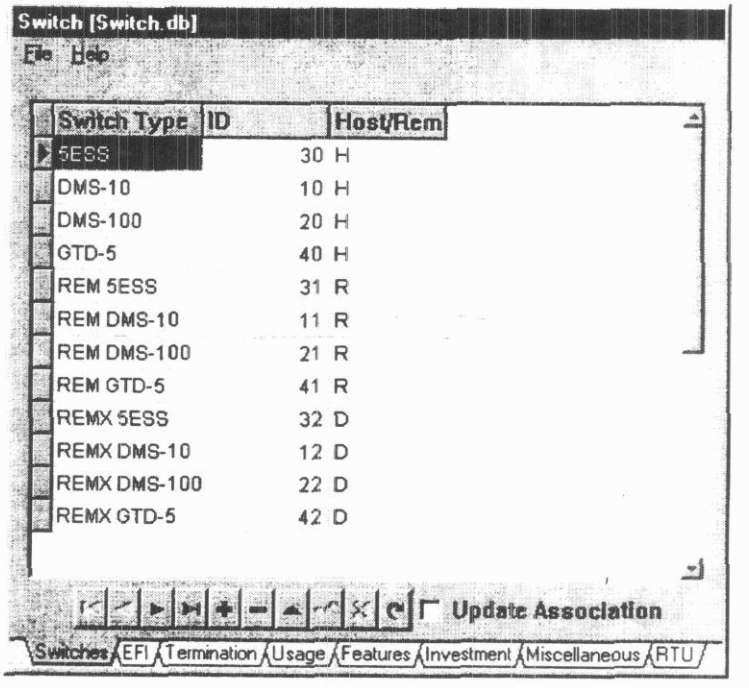
2. Select **File/Exit** to return to the main window.

Viewing Switch Module Data

You can view the switch, wire center, feature, and usage-input data.

Viewing Switch Data

- For information about editing input data, see "Editing Input Data and Mappings" on page 40. For information about the Switch Data, see Appendix A Switch Module Book III within the *ICM Model Methodology package,* or the ICM online help "Input Tables."



Note: The screen shot is for illustrative purposes only.

To view switch data:

1. Select **Table/Switch/Switching**.
   - The system displays the Switch window, which has seven tabs. It displays the Switches tab first, which contains the switch types ICM uses.

SCIS and COSTMOD are switch investment systems that model the investments for the features and functions of switching equipment.

- The EFI tab contains the factors for GTE engineering, furnishing, and installation (EFI) for each switch type and size.
- The Termination tab contains the unit investment outputs from the Switching Cost Information System (SCIS) and COSTMOD for each switch type and line size combination.
- The Usage tab contains the usage unit investment outputs from SCIS and COSTMOD for each switch type and line size combination.
- The Features tab contains the feature unit investment outputs from SCIS and COSTMOD for each switch type and line size combination.
- The Investment/Line tab contains the investment per line calculated from the SCIS and COSTMOD inputs.
- The Miscellaneous tab contains the Investment Adjustment Factor and the investment inputs for Power and 4-Tel test equipment.

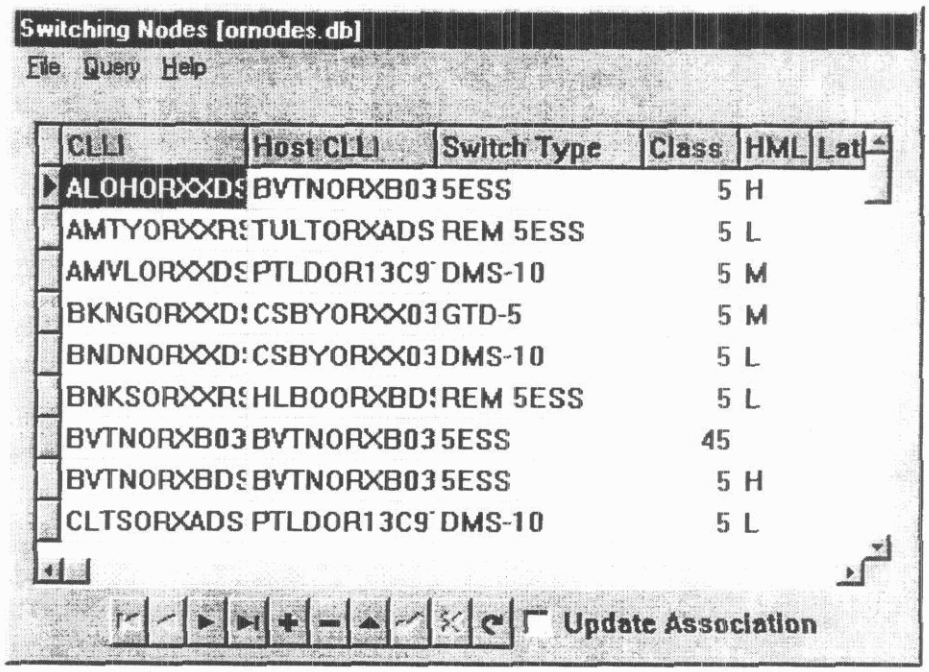
2. Select **File/Exit** to return to the main window.

Viewing Node Data

For information about editing input data, see "Editing Input Data and Mappings" on page 40. For information about the Node data, see Appendix A Switch Module Book III within the *ICM Model Methodology package,* or the ICM online help "Input Tables."

To view node data:

1. Select **Table/Switch/Nodes**.
   - The system displays the Switching Nodes window, that contains the CLLI codes, host CLLI codes for remotes, switch type, switch substitution type, class (4, 4/5, or 5), and total number of lines for each switch.



Note: The screen shot is for illustrative purposes only.

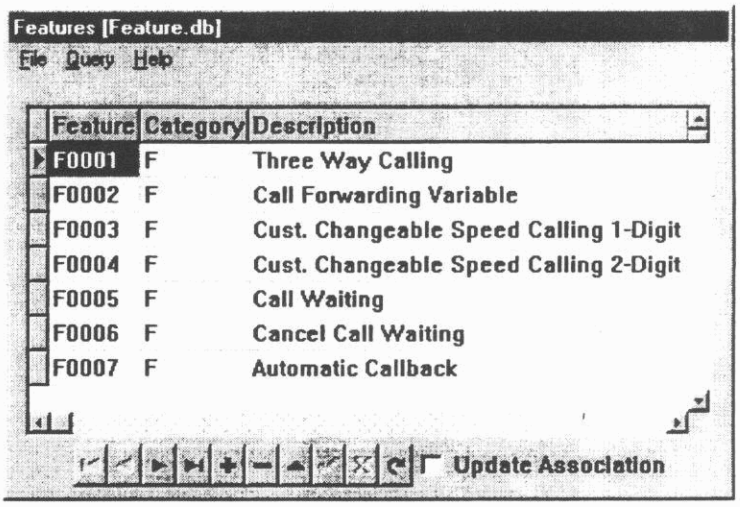
2. Select **File/Exit** to return to the main window.

Viewing Feature Data

For information about editing input data, see "Editing Input Data and Mappings" on page 40. For information about the Switch Features data, see Appendix B Switch Module Book III within the *ICM Model Methodology package,* or the ICM online help "Input Tables."

To view feature data:

1. Select **Table/Switch/Features**.
   - The system displays the Features window, which contains the feature codes, categories, and descriptions.
   - You can now use the query functions to search for information, sort the table information, and sum data. For more information, see "Viewing and Searching Data Tables" on page 28.



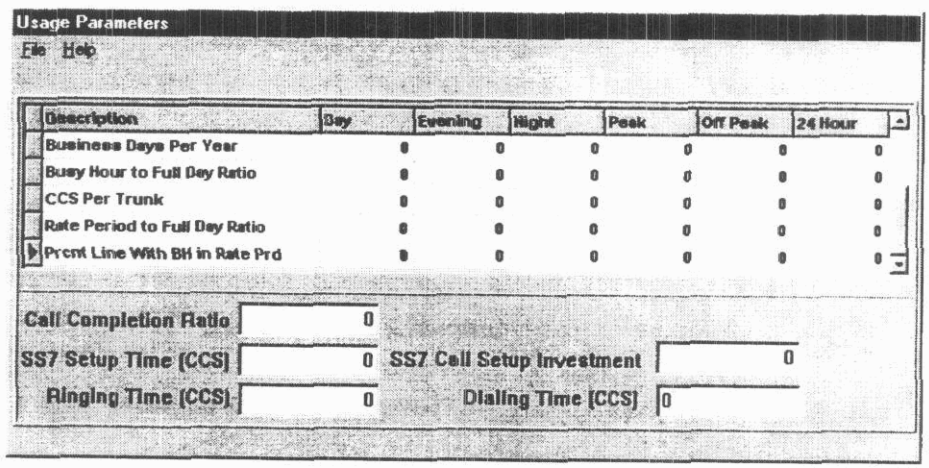
2. Select **File/Exit** to return to the main window.

Viewing Usage Data

For information about editing input data, see "Editing Input Data and Mappings" on page 40. For information about the Usage Data , see Appendix A Switch Module Book III within the *ICM Model Methodology package,* or the ICM online help "Input Tables."

To view usage data:

1. Select **Table/Switch/Usage Parameters**.
   - The system displays the Usage Parameters window that contains the usage information the model uses to convert the unit investments to minutes of use and call set up costs.



Note: The screen shot is for illustrative purposes only.

2. Select **File/Exit** to return to the main window.

Viewing Interoffice Module Data

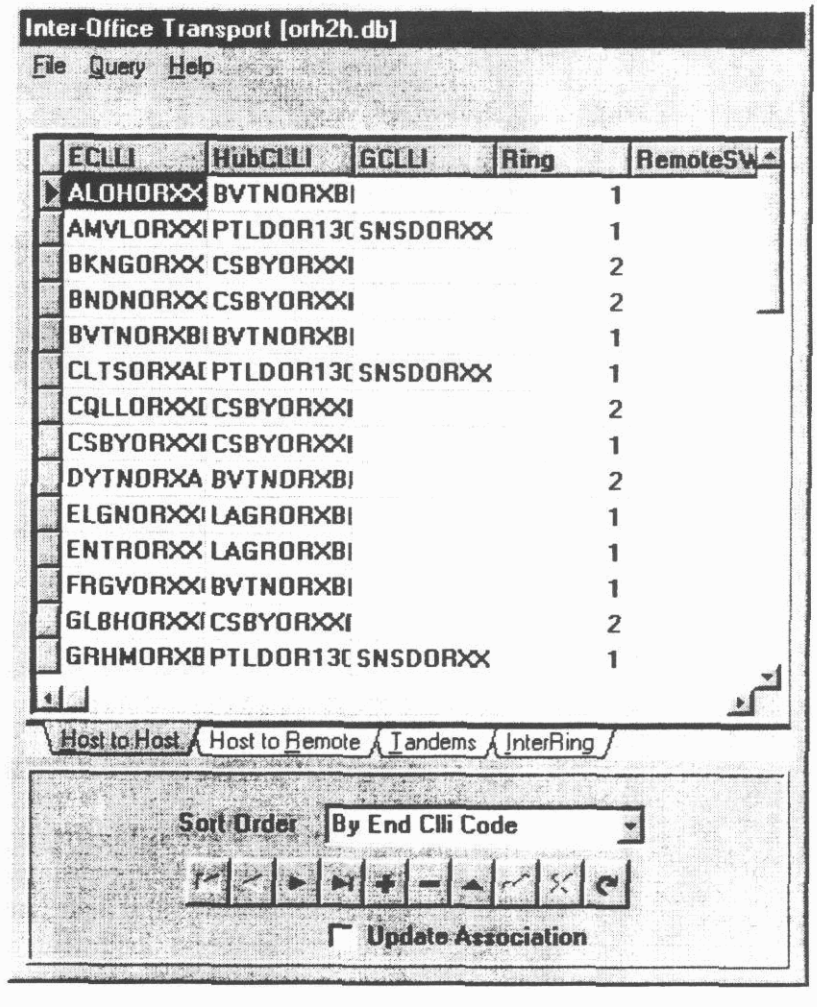
You can view interoffice input data, the outside plant material and placement costs, and assumptions used to calculate facilities sharing, placement, and cable sizing. You can also view the traffic data used to calculate the tandem trunk CCS (centum, or 100, call seconds).

Viewing Interoffice Data

For information about editing input data, see "Editing Input Data and Mappings" on page 40. For information about the Interoffice data , see Appendix A Interoffice Transport Module Book IV within the *ICM Model Methodology package,* or the ICM online help "Input Tables."

To view interoffice data:

1. Select **Table/Interoffice/Interoffice**.
   - The system displays the Interoffice Transport window that has two tabs. The first tab, Host to Host, contains CLLI codes and other data for each end office and its associated tandem.



Note: The screen shot is for illustrative purposes only.

- The Remote to Host tab contains the CLLI codes and other information about each host and its associated remotes.
- To print the contents of the current tab, select **File/Print**.
- You can now use the query functions to search for information, sort the table information, and sum data. For more information, see "Viewing and Searching Data Tables" on page 28.

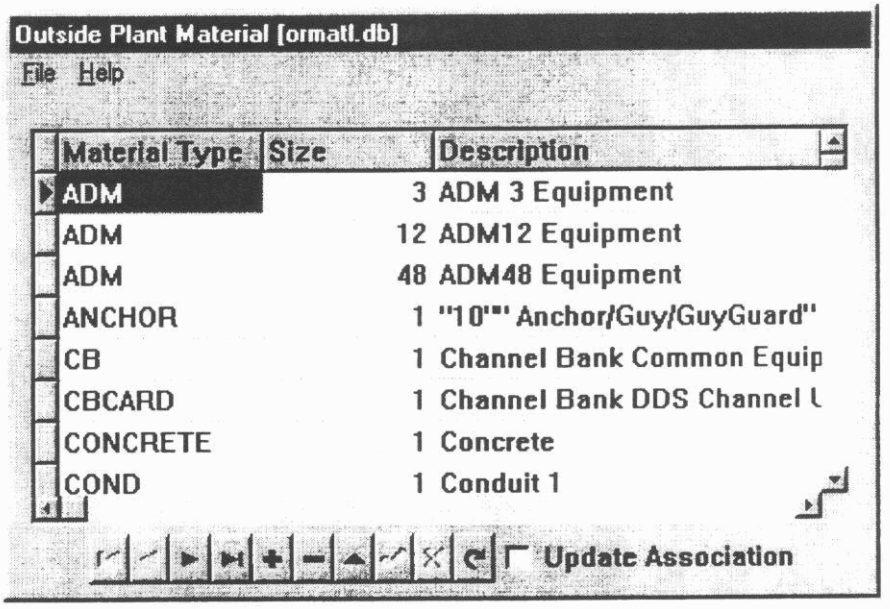
2. Select **File/Exit** to return to the main window.

Viewing Outside Plant Material or Placement Costs (Interoffice)

For information about editing input data, see "Editing Input Data and Mappings" on page 40. For information about the Outside Plant Material or Placement Costs of the Interoffice Module, see Appendix A Interoffice Transport Module Book IV within the *ICM Model Methodology package,* or the ICM online help "Input Tables."

To view outside plant material costs or placement costs:

1. Select **Table/Interoffice/Materials or Table/Interoffice/Placement Costs**.
   - Below is the Outside Plant Material table.



Note: The screen shot is for illustrative purposes only.

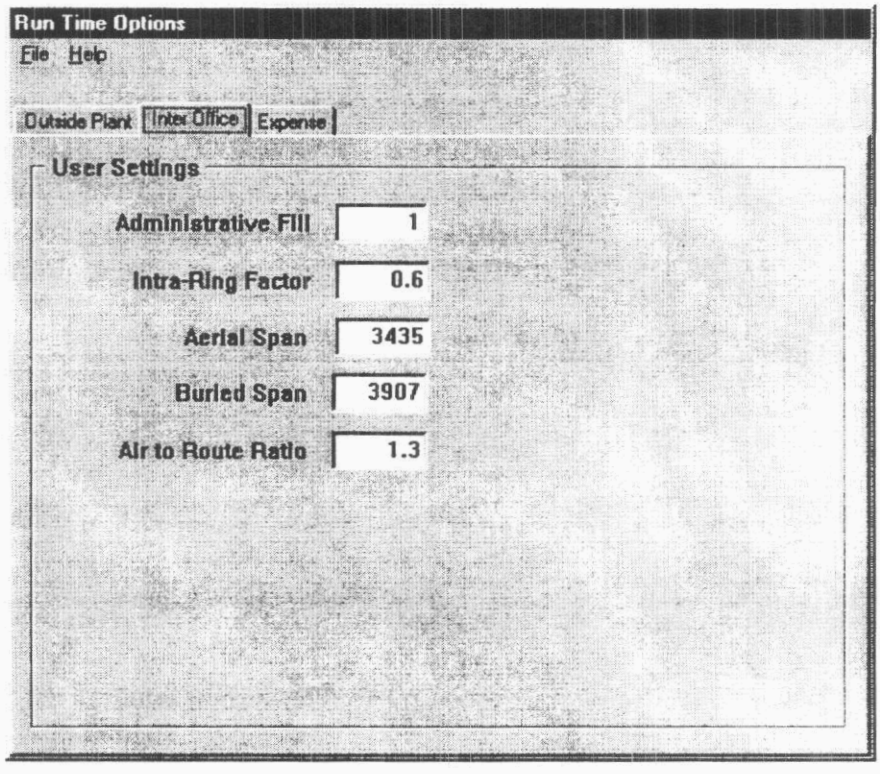
   - To print the contents of the current table, select **File/Print**.
2. Select **File/Exit** to return to the main window.

Viewing Interoffice User Options

For information about editing input data, see "Editing Input Data and Mappings" on page 40. For information about the Interoffice User Options data, see Appendix A, Conceptual Framework (Book I) within the *ICM Model Methodology package,* or the ICM online help "Input Tables."

To view the interoffice user options:

1. Select **Options/User**.
   - The system displays the Run Time Options window.
2. Click the Interoffice tab.



Note: The screen shot is for illustrative purposes only.

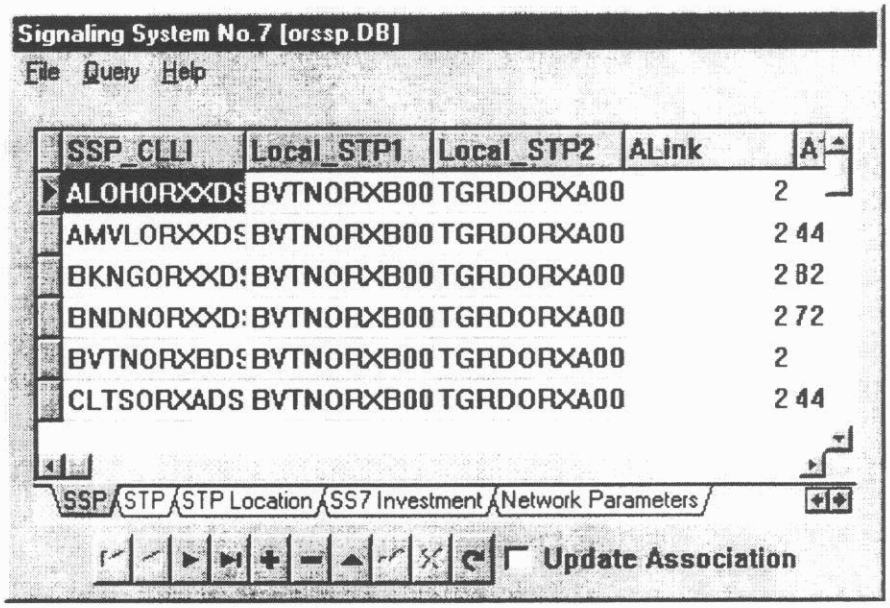
   - To print the User Options, select **File/Print**.
3. Select **File/Exit** to return to the main window.

Viewing Signaling System 7 (SS7) Module Data

For information about editing input data, see "Editing Input Data and Mappings" on page 40. For information about the SS7 data see Appendix A SS7 Module Book V within the *ICM Model Methodology package,* or the ICM online help "Input Tables."

To view SS7 data:

1. Select **Table/SS7**.



Note: The screen shot is for illustrative purposes only.

- The system displays the Signaling System No. 7 window. The window is divided in six tabs – Service Switching Point (SSP) tab, Signal Transfer Point (STP) tab, Signal Transfer Point Location (STP Location) tab, SS7 Investment tab, Network Parameters tab and Loading Factors. Description of each SS7 sub-tabs is as follows:
- SSP provides CLLI codes of the host/stand-alone offices in the state, along with the CLLI codes of STPs supporting the SSPs. As part of a forward-looking model, each host/stand-alone office is assumed to be SSP-equipped.

- STP provides general data about STPs, such as geographic coordinates, port quantities, and expenses for links to other STPs.
- STP Location for each regional STP, indicates where each pair of service-specific SCPs is located.
- SS7 Investment provides SS7 hardware and software investment by network component, along with estimated labor hours.
- Network Parameters provides SS7 network engineering and utilization data required for cost calculations.
- Loading Factors provides state-specific minor material loadings and labor rates to convert material investments to total investments.
- You can use the query functions to search for information, sort the table information, and sum data. For more information, see "Viewing and Searching Data Tables" on page 28.

2. Select **File/Exit** to return to the main window.

Viewing Expense Module Data

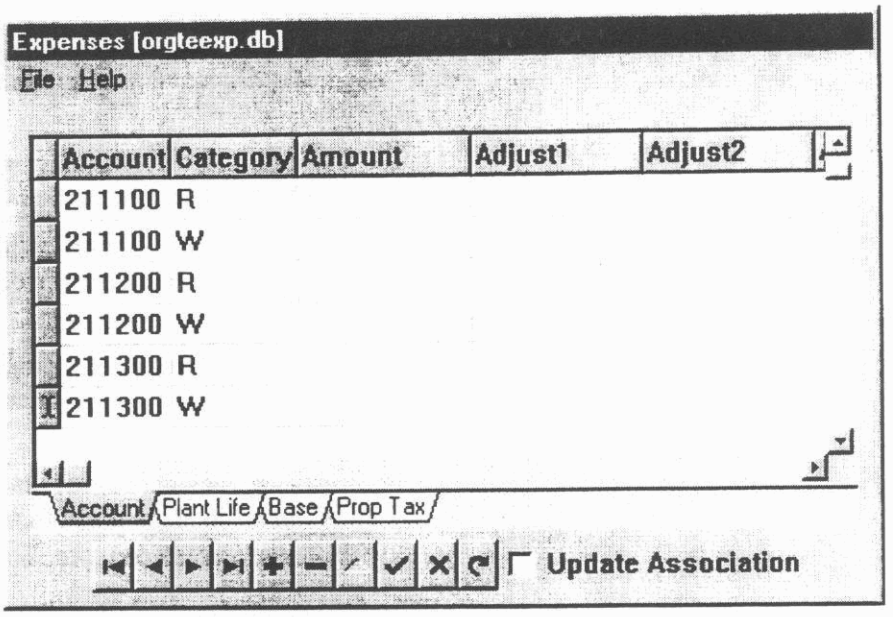
You can view the Expense Module investment factors and activity based costs.

Viewing Investment Factors

For information about editing input data, see "Editing Input Data and Mappings" on page 40. For information about the Expense Module data, see Appendix A Expense Module Book VI within the *ICM Model Methodology package*, or the ICM online help "Input Tables."

To view investment factors data:

1. Select **Table/Expense/Investment Factors**.
   - The system displays the Expenses window with four tabs. The first tab, Account, contains fractions of general ledger accounts mapped to various cost pools.



Note: The screen shot is for illustrative purposes only.

- The Plant Life tab contains lives and salvage values of plant by account.
- The Base tab contains the company's capital structure and tax rates.
- The Prop Tax tab identifies whether property taxes are to be calculated for general support assets, by account.

2. Select **File/Exit** to return to the main window.

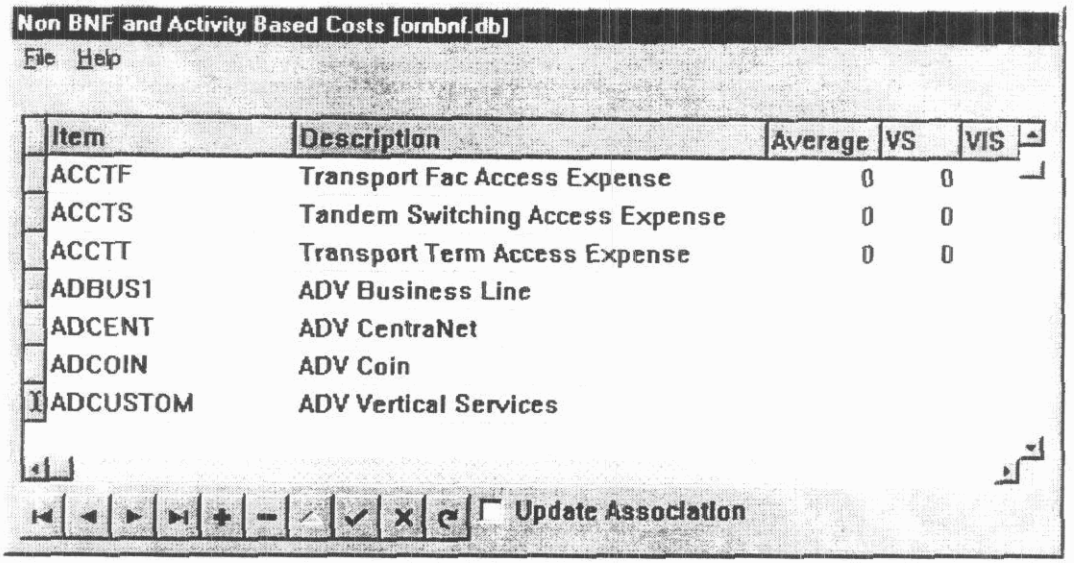
Viewing Non BNF and Activity Based Costs

For information about editing input data, see "Editing Input Data and Mappings" on page 40. For information about the Activity Based Costs data, see Appendix A Expense Module Book VI within the *ICM Model Methodology package,* or the ICM online help "Input Tables."

To view activity-based costs:

1. Select **Table/Expense/Non BNF and Activity Based Cost**.

- The system displays the Non BNF and Activity Based Cost window and the activity based costs.



Note: The screen shot is for illustrative purposes only.

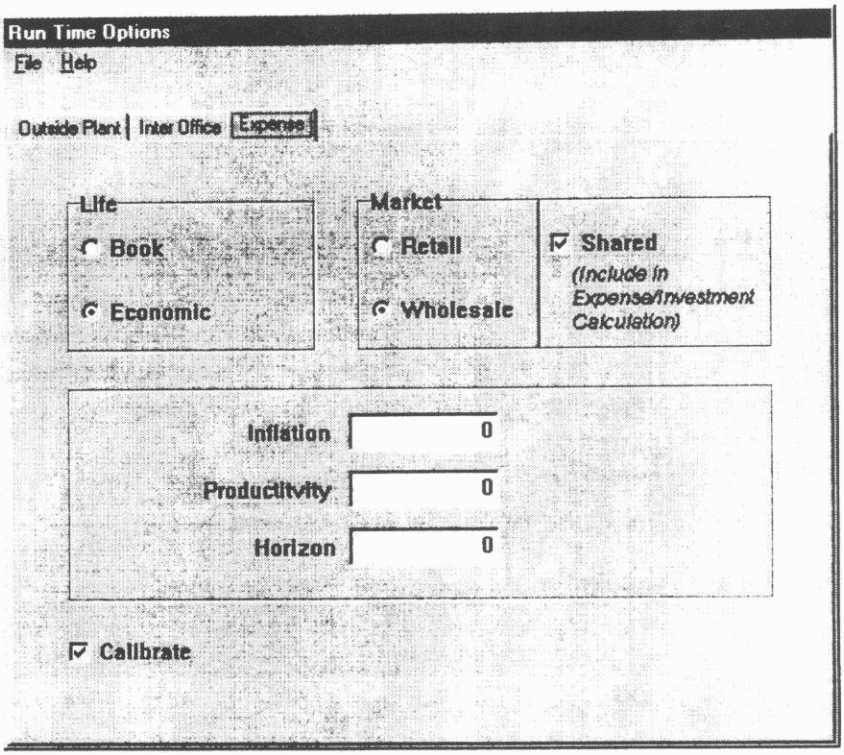
2. Select **File/Exit** to return to the main window.

Viewing Other Expense Inputs

For information about editing input data, see "Editing Input Data and Mappings" on page 40. For information about Other Expense Input data see Appendix A Preface/ICM Overview Book I within the *ICM Model Methodology package*, or the ICM online help "Input Tables."

To view the other expense inputs:

1. Select **Options/User**.
   - The system displays the Run Time Options window.
2. Click the Expense tab.



Note: The screen shot is for illustrative purposes only.

- The Inflation, Productivity, and Horizon fields are not currently used.

3. Select **File/Exit** to return to the main window.

Viewing and Searching Data Tables

You can view the input and intermediate data tables ICM creates during processing from the Query window.

For information about editing input data, see "Editing Input Data and Mappings" on page 40. For information about the data, see Appendix A of each module (Book I-VII) within the *ICM Model Methodology package*, or the ICM online help "Input Tables."

From the Query window, you can perform several functions related to the table data:

- View table data
- Search for information
- Save search results
- Select how the columns sort
- Sum a single numerical column
- Find the highest value in a numerical column

Viewing Table Data

You can view the input and intermediate data tables ICM creates during processing from the Query window.

Below are the file names of the intermediate data tables:

File Name	Contents
ACTCMPNT.DB	Investment and unit quantities by CLLI for all components.
ACTDIST.DB	Distribution investments and units by CLLI and components.
ACTFEED.DB	Feeder investments and unit by CLLI and components.
ACTH2H.DB	Host to host interoffice transport investments and unit by CLLI and component.
ACTH2R.DB	Host to remote interoffice transport investments and unit by CLLI and component.
ACTSS7.DB	SS7 investments and unit by CLLI and component.
ACTSWTCH.DB	Switch investments by CLLI and component.
EXPRSLT	Expense factors by account.

To view any data table:

1. Select **Utilities/Table Viewer**.
   - The system displays the Query window.
2. Select **File/Open Table**.
   - The system displays the Open window.
3. Select the data table you want.
4. Click **Open**.
   - The system displays the table you selected.

- You can use the query functions to search for information, sort the table information, and sum data. For more information, see "Viewing and Searching Data Tables" on page 28.

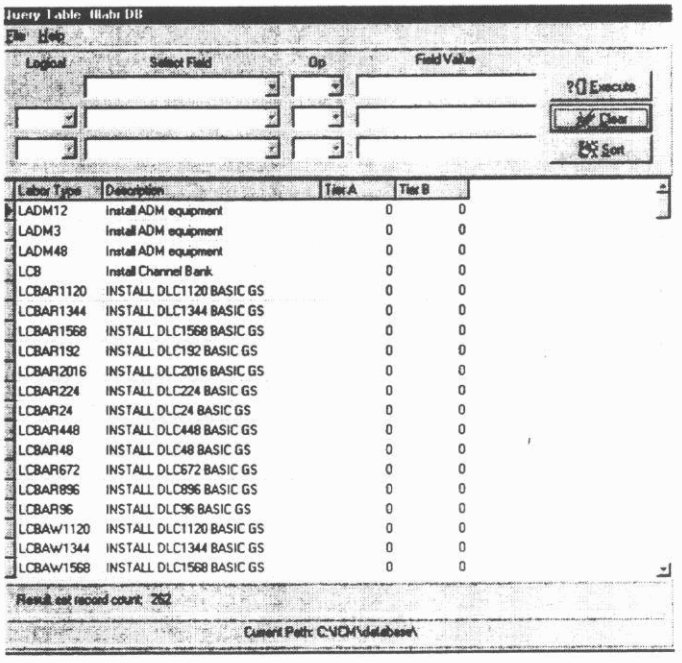
5. Select **File/Exit** to return to the main window.

Searching for Table Information

You can search for information in a table using the query function in the Table Viewer. If you are familiar with how to create an SQL statement, see "Entering an SQL Statement" on page 32.

To search for information:

1. Open a table.
   - For more information, see "Viewing Table Data" on page 29.
   - The system displays the Query Table window.



Note: The screen shot is for illustrative purposes only.

2. Select a field you want to search on from the Select Field drop-down list.
3. Select an operator to use for your search from the Op drop-down list.
   - The available operators are: = (equal to), < > (not equal to), < (less than), > (greater than), Like, and Not Like.
   - The Like operator searches for the characters you enter. You can use the % character as a wildcard that matches one or more characters. For example, if you search for REM% as the Switch Type, the system finds all rows in which the Switch Type starts with REM. You can use the % wildcard at the beginning, at the end, or on both sides of the characters you want to find.
   - The Not Like operator searches for rows that do not contain the characters you enter. You can also use the % wildcard with Not Like searches. For example, if you use Not Like and search for REM% as the Switch Type, the system finds all rows in which the Switch Type does **not** start with REM.
4. Enter what you want to search for in the Field Value field.
   - You must enter the value exactly as it is in the table.
5. Click **Execute**.
   - The system displays the information that matches your search.
   - You can change the search criteria, or enter additional search criteria in the fields below the first row of search criteria. If you enter additional search criteria, you must select a Logical operator. Use "And" if you want to find information that contains all of the field values you enter. Use "Or" if you want to find information that contains any of the values you enter.
   - Click **Execute** to see the new search results each time you change the search criteria.

- You can save your search results to a table or a text file. For more information, see "Saving Search Results" on page 32.

Entering an SQL Statement

If you are familiar with how to create an SQL statement, you can enter a statement directly.

1. From the Query window, select **File/Enter Sql Statement**.
   - The Query Statement window displays.
2. Enter an Interbase SQL statement.
   - This is the Borland syntax for SQL statements.
3. Click **Execute**.
   - You can save your search results to a table or a text file. For more information, see "Saving Search Results" on page 32.

Saving Search Results

Once you search for information in the Query window, you can save your results to another table or to a text file.

To save results to a table or text file:

To save to a table	To save to a text file
1. From the Query window, select **File/Build Result Table**. □ The "Enter new table name" window displays.	1. From the Query window, selec **File/Export to text file**. □ The "Enter new table name" window displays.
2. Enter a file name.	2. Enter a file name.
3. Click **Save**.	3. Click **Save**.

Selecting How the Columns Sort

You can select how the columns sort information in a table using the query function in the Table Viewer. The sort function sorts the data in the columns you select in alphanumeric order.

To select how the columns sort:

1. Open a table.
   - For more information, see "Viewing and Searching Data Tables" on page 28.
   - The system displays the Query Table window.
2. Click **Sort**.
   - The system displays the Select Sort Order window.
3. Click the column name in the Field List that you want to be first column sorted.
4. Continue clicking column names from the Field List.
5. Click **OK**.
   - The system returns to the Table Viewer window and displays the data sorted according to your selections.

Summing a Single Column

You can sum a single numerical column using the query function in the Table Viewer.

To sum a single column:

1. Open a table.
   - For more information, see "Viewing and Searching Data Tables" on page 28.
   - The system displays the Query Table window.
2. Click any cell in a numerical column.
3. Right-click.
4. Select **Sum this column** from the pop-up menu.
   - The system sums the column and displays the total at the bottom of the window. In a large file, the process may take a few minutes.

Finding the Highest Value in a Column

You can find the highest value in a column using the query function in the Table Viewer.

To find the highest value in a column:

1. Open a table.
   - For more information, see "Viewing and Searching Data Tables" on page 28.
   - The system displays the Query Table window.
2. Click any cell in a numerical column.
3. Right-click.
4. Select **Find max column value** from the pop-up menu.
   - The system highlights the highest value. In a large file, the process may take a few minutes.

Viewing Demand Information, Cluster and Feeder Route to Digital Loop Carrier (DLC).

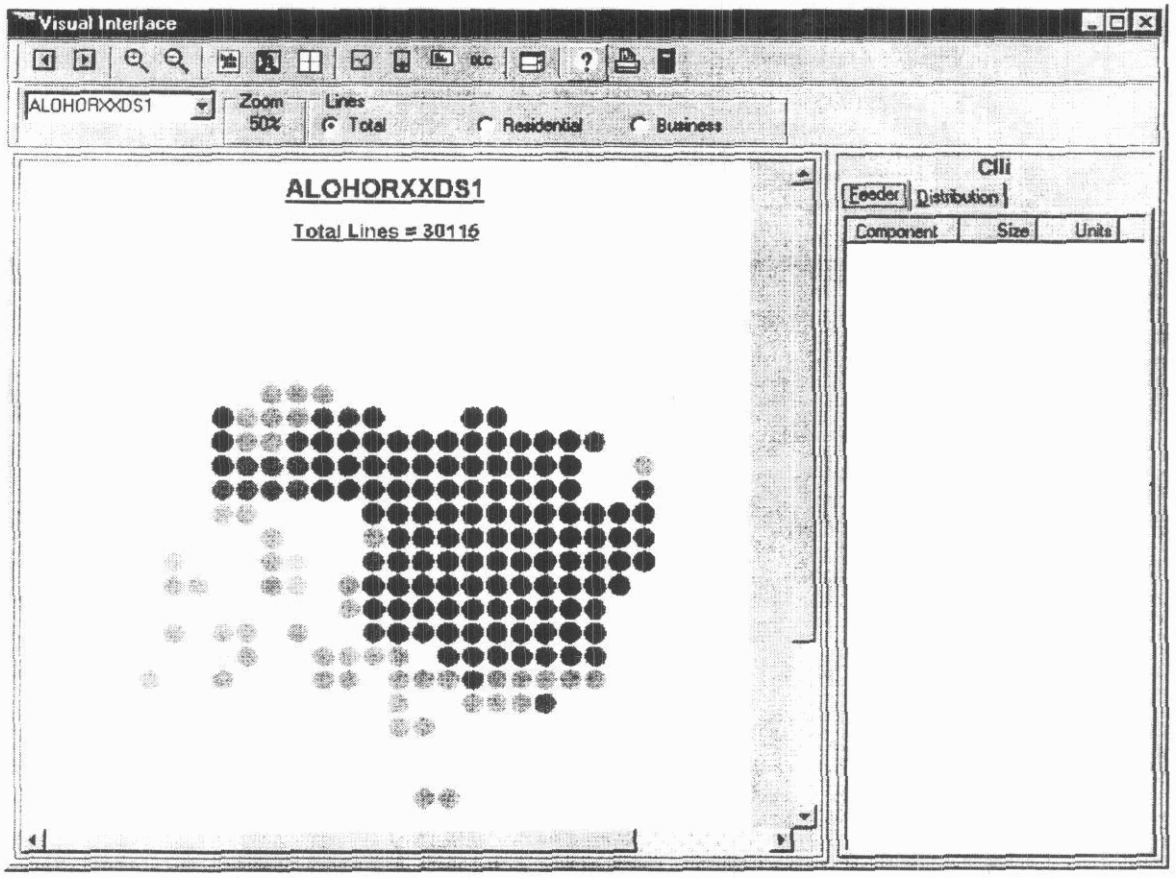
You can view Demand Information, Cluster and Feeder route for each CLLI in a wire center.

1. Click **Visual Interface** on the menu bar.

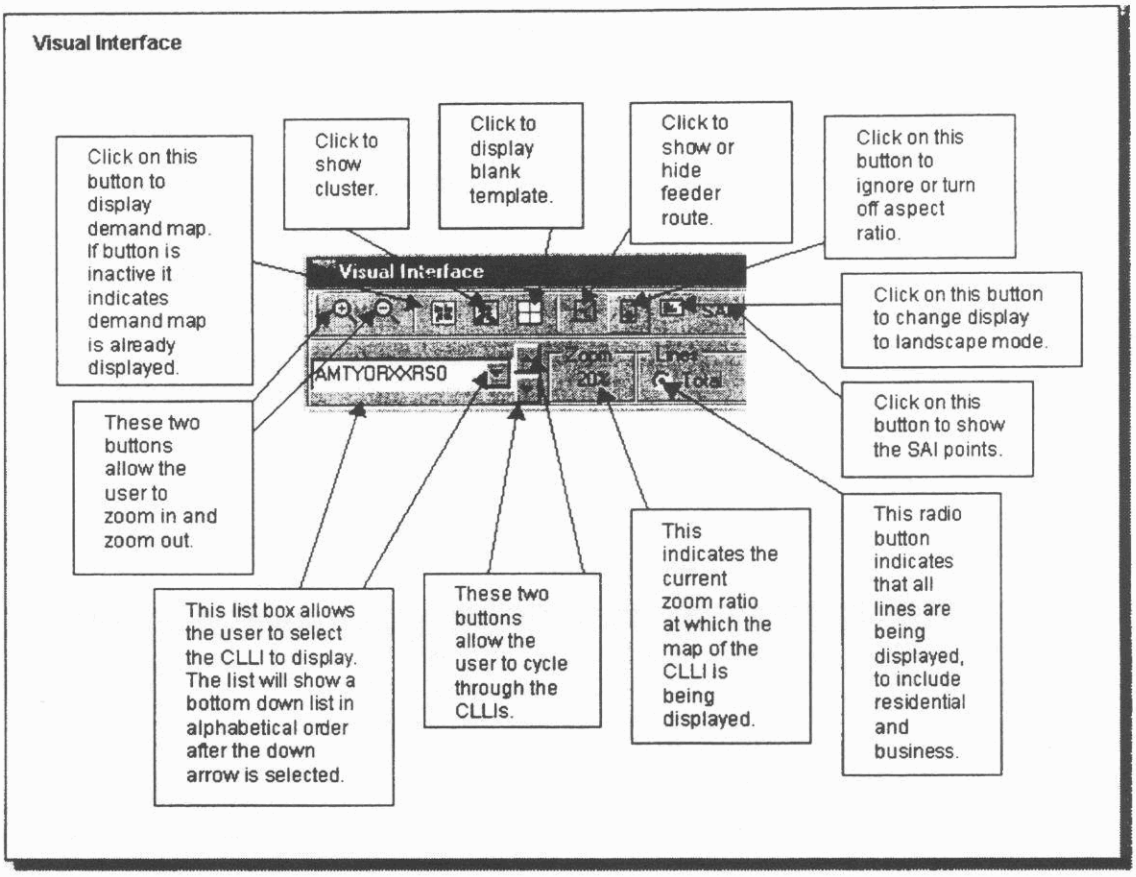
   The following diagrams will walk you step-by-step through process of what to expect after you click **Visual Interface** on the menu bar.

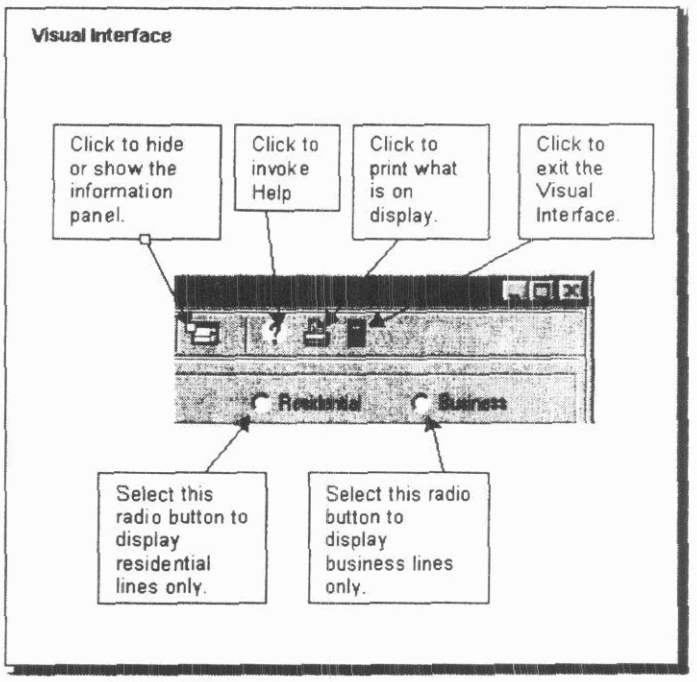
   To understand the process better, please review the following three diagrams of Visual Interface screen together.

Click anywhere on the image for information on any of the controls



Note: The screen shot is for illustrative purposes only.


Visual Interface
Click on this button to display demand map. If button is inactive it indicates demand map is already displayed.
Click to show cluster.
Click to display blank template.
Click to show or hide feeder route.
Click on this button to ignore or turn off aspect ratio.
Visual Interface
Click on this button to change display to landscape mode.
AMTYORXXRS0
Zoom 202
Total
These two buttons allow the user to zoom in and zoom out.
Click on this button to show the SAI points.
This radio button indicates that all lines are being displayed, to include residential and business.
This indicates the current zoom ratio at which the map of the CLLI is being displayed.
This list box allows the user to select the CLLI to display. The list will show a bottom down list in alphabetical order after the down arrow is selected.
These two buttons allow the user to cycle through the CLLIs.

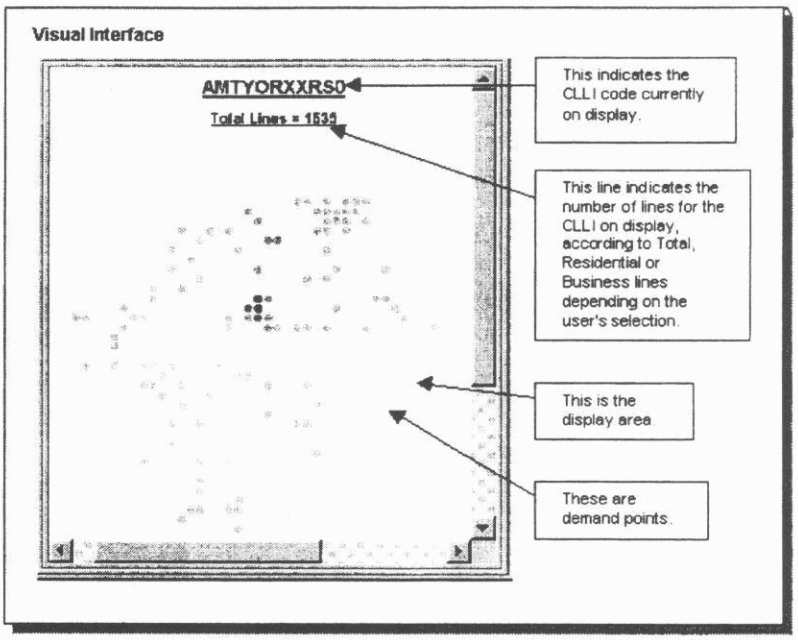


Note: The screen shots are for illustrative purposes only.

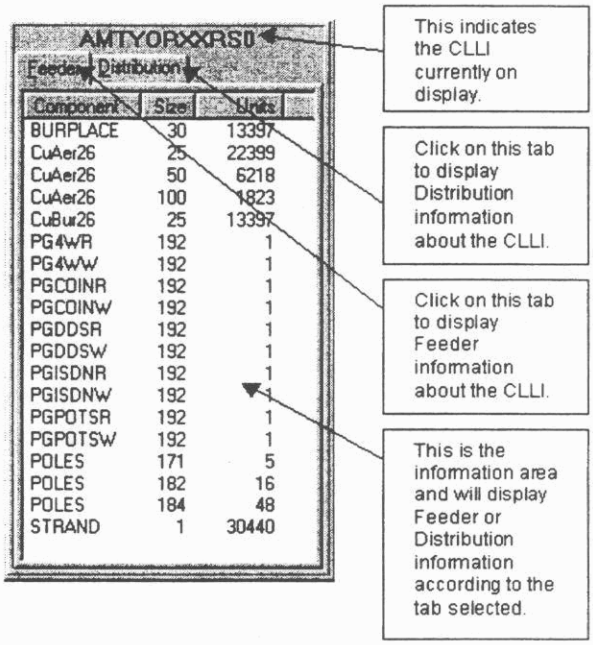
The first diagram above shows Visual Interface screen in full display and the second and third diagrams explains in detail the Visual Interface screen's tool bar and radio buttons.

2. The following two diagrams divide the first diagram (showing Visual Interface screen in full display) in two parts and provide a detail explanation of each part: Display Area and Information Panel.

Visual Interface



Visual Interface



Note: The screen shots are for illustrative purposes only.

The diagrams above give a graphic overview of the Visual Interface screen.

Viewing Source Code

You can view the source code for each ICM Module from the CD ROM. It does not transfer to your hard drive during installation.

To view the source code:

1. Use the Windows Explorer to navigate within the ICM CD ROM.
2. Go to the `Source` directory.
    - The files in this directory are ASCII files that contain all of the source code. You can use any text editor (MS Word, Notepad, or other program) to view or print the source code.

For more information about the system files, refer to the *ICM System Manual.*

Editing Input Data and Mappings

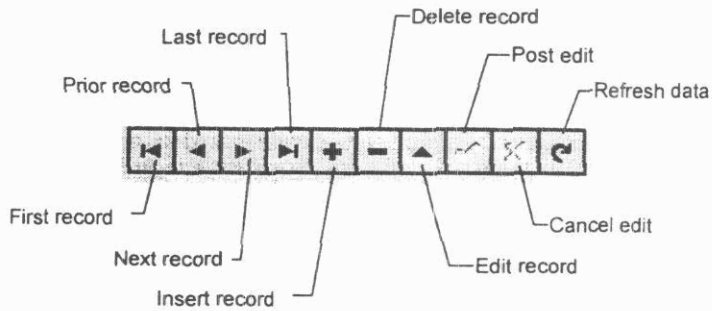
Editing Data

You can edit the data in the Run Time Options screen. For information about the data in each table, see the *ICM Model Methodology*, Appendix A or the ICM online help "Input Tables."

Editing Data Tables

You can edit the data in any of the tables under the Table menu. The editing process is the same for each table.

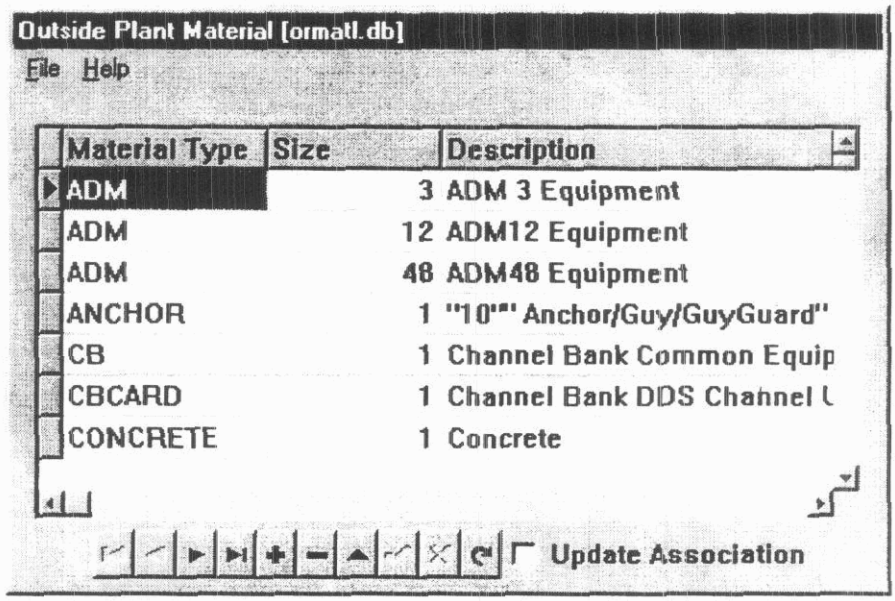
You use the editing tools to move through the table and edit the data. If you place the mouse pointer over a button, the system displays the button's name:



The steps below use the Outside Plant Material table as the example.

To edit data tables:

1. Display the table you want to edit.
   - In this example, we select **Table/Loops/Material**.



2. Use your mouse or keyboard to move around and edit.

To use your mouse	To use your keyboard
1. Click the cell you want to edit. 2. Click the cell again. 3. Enter the new data. 4. Click **Post Edit**. □ The system saves your changes. If you receive a Key Violation error, you have created a duplicate record. The system can only store one record for each set of key fields in a table. You must change the duplicate record and make it unique.	1. Press **Tab** until you reach the cell you want to edit. 2. Press **Enter**. 3. Enter the new data. 4. Tab to the next row. □ The system saves your changes. If you receive a Key Violation error, you have created a duplicate record. The system can only store one record for each set of key fields in a table. You must change the duplicate record and make it unique.

- Use the First record, Prior record, Next record, and Last record buttons to move to the row of data you want.
- To insert a new row of data, click Insert record. To delete a row of data, click **Delete record**.

3. Select **File/Exit** to return to the main window.

Importing Data Files

You can import comma-delimited data into any of the tables under the Table menu to replace the existing data. The importing process is the same for each table.

Note: Before importing data, you must create an ASCII file containing the data you want. You must be sure that the data is in the same order as in the tables in ICM and is comma delimited.

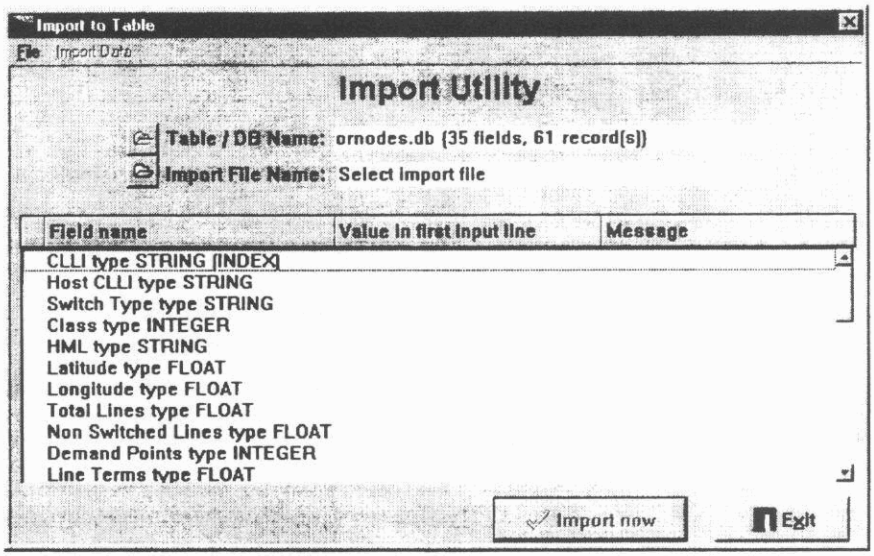
The import utility helps you ensure that the data in the file is correct and gives you error information both before and after importing the file. You can also select your import option:

- Replace the current data in the file
- Add new records without replacing duplicate records
- Add new records and replace duplicate records

Note: If you want to save the data currently in the table, you should export the data to a file before importing. For more information, see "Exporting Specific Input Data Tables" on page 47.

To import a data file:

1. Display the table you want to replace.
2. Select **File/Import**.
   - The system displays the Import Utility window showing the name of the current file and how many fields and records the file contains. The window also lists each field name in the current table and its type.



3. Click the folder next to Import File Name.
   - The Open window displays.
4. Select the file you want to import.
5. Click **Open**.
   - The system returns to the Import Utility window and displays the values from the first line in your file.
   - The window also displays a green, yellow, or red light on the left side of the window. A yellow or red light means that the data in the first line of your file has a problem. Read the information in the Message column to find out what the problem is.
   - You can continue your import without correcting any problems, but your data may not import properly.
6. Click **Import now**.
   - The import options window displays.
7. Select the import option you want.
8. Click **Continue**.
   - The system imports the data and displays information about the import.

- If there are errors, an information message gives you the name of the file that contains each record that was not processed. You can then correct the errors in the records and import just those records using the Add option.
- If the system displays a floating decimal error, you should check the input. The error means that the value is not a floating point number. The data may contain a comma instead of a decimal, or a letter instead of a number.

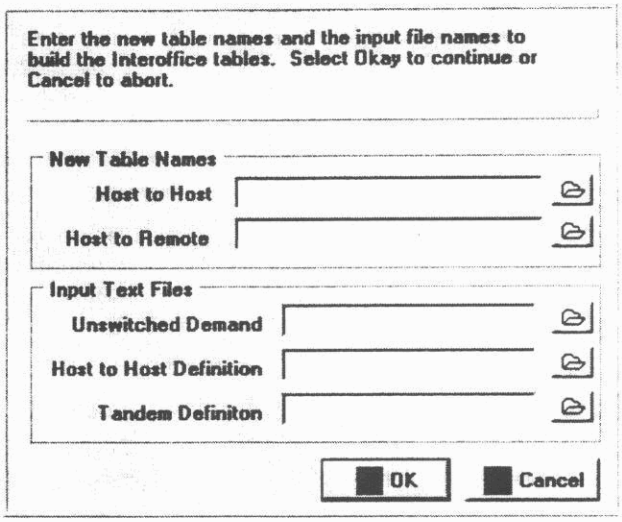
9. Click **OK**.
   - The system returns to the table window and displays your new data. If there were errors, the system asks if you want to display the log file, which contains a running history about the import process.
10. Select **File/Exit** to return to the main window.

Building or Rebuilding the Interoffice Tables

You can build new or rebuild the existing interoffice tables: Host to Host and Host to Remote. Use this function to import data into these tables.

To build or rebuild the interoffice tables:

1. Select **Table/Interoffice/Interoffice**.
   - The Interoffice Transport window displays.
2. Select **File/Build Tables**.
   - The window below displays.



3. In the New Table Names section, enter the names of Host to Host and Host to Remote tables that you want to create or rebuild.
   - Instead of entering the file names, you can click the folder button and navigate to the file locations.
   - These are the tables ICM uses to process transport data.
4. In the Input Text Files section, enter the names of the files you want to use to create or rebuild the interoffice tables.
   - Instead of entering the file names, you can click the folder button and navigate to the file locations.
5. Click **OK**.

Managing Multiple Sets of Data Files

You can have multiple data files for each data table in ICM. You can change one or more of the data files that ICM uses at any time.

Note: Before changing to a different data file, you must import the data into ICM using the steps in the previous section.

Replacing a Single Data File

To replace a single data file:

1. Display the table you want to replace.
2. Select **File/Open**.
   - The system displays the Open window.
3. Select the data file you want to use.
4. Click **OPEN**.
   - The system displays the table with the data file you selected.
5. Click the Update Association checkbox to associate the new file with ICM.
6. Select **File/Exit** to return to the main window.

Replacing One or More Data Files

To replace one or more data files:

1. Select **Table/Associate**.
   - The system displays the Associate window that lists each data table in the model and the full path of the current data file.
2. Select the Lookup Table you want to replace.
3. Click **Browse**.
   - The system displays the Use window.
4. Navigate to the directory that has the data file you want to use and select the file.
5. Click **OPEN**.
   - The system returns to the Associate window.
6. Repeat steps 2 through 5 for each table you want to replace.
7. Click **OK** to return to the main window.

Exporting Specific Input Data Tables

You can export any of the ICM input data tables to a comma-delimited ASCII file. After you export the file, you can edit it and then import the revised file back into ICM.

To export a data table:

1. Display the table you want to export.
   - For more information, see "Viewing Input Data" on page 9 or "Viewing and Searching Data Tables" on page 28.
2. Select **File/Export**.
   - The system displays the Save As window.
3. Enter a file name.
4. Click **SAVE**.
   - The system saves the data in a comma-delimited ASCII file.
5. Select **File/Exit** to return to the main window.

Editing Run Time Options

You can edit the Run Time Options, such as outside plant sharing percent, that are under the Options menu.

To edit the Run Time Options:

1. Select **Options/User**.
   - The system displays the Run Time Options window.
2. Go the tab you want to edit.
3. Press **Tab** to go to the field you want to edit.
4. Enter the new data.
5. Press **Tab**.
6. Select **File/Exit** to return to the main window.

Editing the Mappings

Products and Services mappings are not applicable in states where results for products and services are not provided.

You can edit the mappings for Products and Services, Unbundled Network Elements (UNEs), Basic Network Functions (BNFs), and Universal Service Fund (USF). You can also edit the mapping groups and the product and service groups.

Editing the Product Mapping

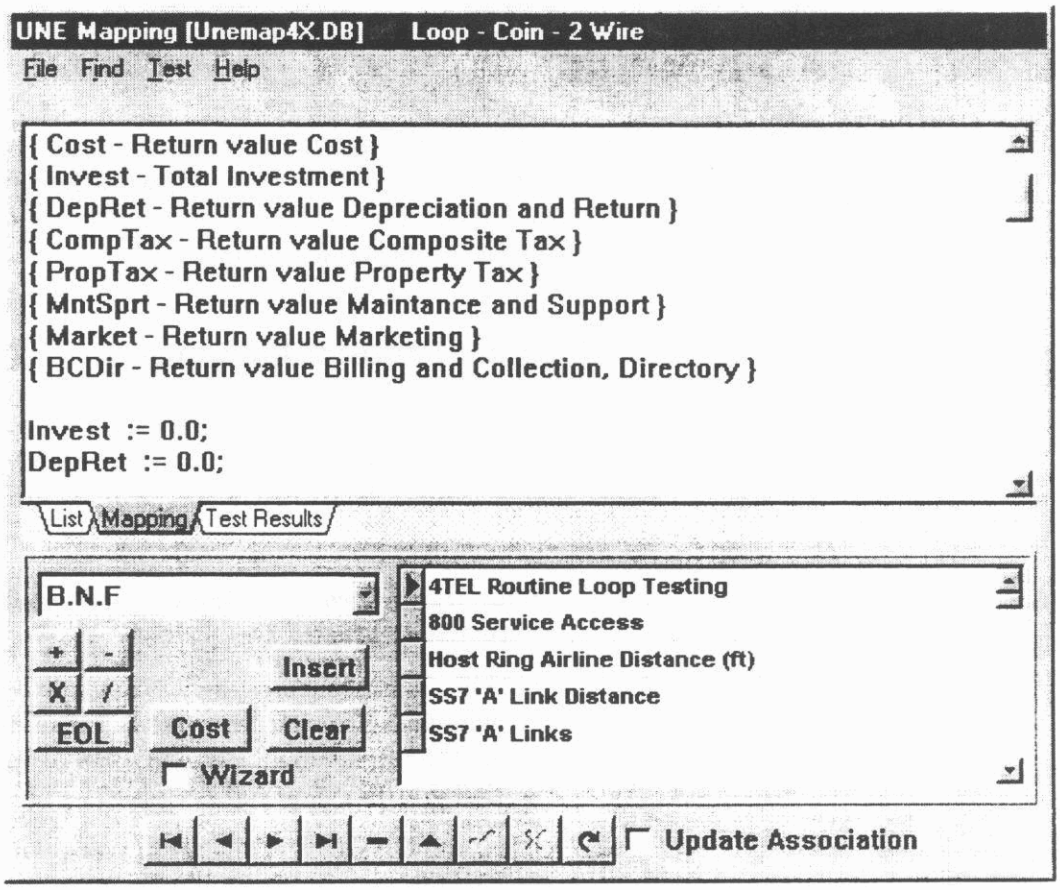
You can edit the mappings for Products and Services, UNEs, BNFs, and USF. You edit all of the mappings the same way. The steps below use UNEs as an example.

To edit element mapping:

1. Select **Table/Product Mapping/Unbundled Network Elements**.
   - The system displays the UNE Mapping window which has three tabs. It displays the List tab first which contains a list of the UNEs that you can view and define.
2. Select the UNE that you want to work with from the List tab.
   - The List tab lets you control the format of the UNE report. You can change the Sequence in

which the UNEs appear and the Indent (in spaces) of each UNE. If you enter False in the Label column, the system generates the costs for that UNE. If you enter True, the UNE is shown as a label or heading. The Format column controls the format of the number. For example, %10.2F creates 10 columns with two decimal places and floating point decimal.

3. Click the Mapping tab.
   - The top portion of the Mapping tab shows how GTE maps the unit investments and expenses to the UNE you selected.
   - You can delete mapping components by selecting them and pressing **Delete**.
   - If you want to add components to the mapping, continue with step 4 below.



4. Select the type of component you want to add from the drop-down list.
5. Select the component you want to add from the list next to the drop-down list.
6. If you are editing an existing mapping, cut the line that starts with: cost=.
   - Cutting the cost line copies the line to your Clipboard so that you can paste it back in at the end. If you do not remove the cost line, the new data is added below the cost line and will not be part of the cost calculation.
7. To map the investment and expenses for each component in one step, turn on the Wizard.
   - Click next to the Wizard to turn it on. If you see a checkmark next to the Wizard, it is on.
   - If you do not turn on the Wizard, you must add each investment and expense related to the component separately.
8. Click **Insert** to add the component you selected to the UNE mapping.
   - The system adds all of the appropriate investments and expenses.
   - In the top portion of the window, the system displays in brackets the abbreviation and definition for each item it added. Items in the brackets are just comments that are not part of the mapping code.
9. Repeat steps 4 through 7 for each component that you want to map to the UNE.
   - Because the loop and transport network components available in BNF mapping are total investments, these components must be unitized by dividing by the appropriate unit, such as total lines or total distance, to arrive at a per unit investment. Unitization is not required for the switching and SS7 components because those components enter the tables as investment per unit.
10. Paste the cost line back in if you cut it in step 6 or click **Cost** to generate the code to calculate the monthly cost.

- If the cost unit for the UNE you are mapping is minutes or messages, you should delete the "/ 12" in the cost line.
- For more information about the coding rules for the mapping, see "Appendix A: Programming Language Alpha (PL-A)" on page 57.

11. Enter a weight value to determine the appropriate method of calculating a weighted average for the aggregation from wire center to group level and then to state level.
    - You must enter Weights for BNF, UNE, and product and services mapping. You can insert them anywhere in the mapping. For example, Weight:=.@TOTLINE[1];
12. To save your changes, click **Post Edit** on the editing tools.
13. Click **Test**.
    - This tests the code to be sure that you have not changed the syntax to something that the compiler cannot use. The numbers that display on the Test Results tab do not reflect actual data.
14. Repeat steps 2 through 13 for each UNE you want to generate.

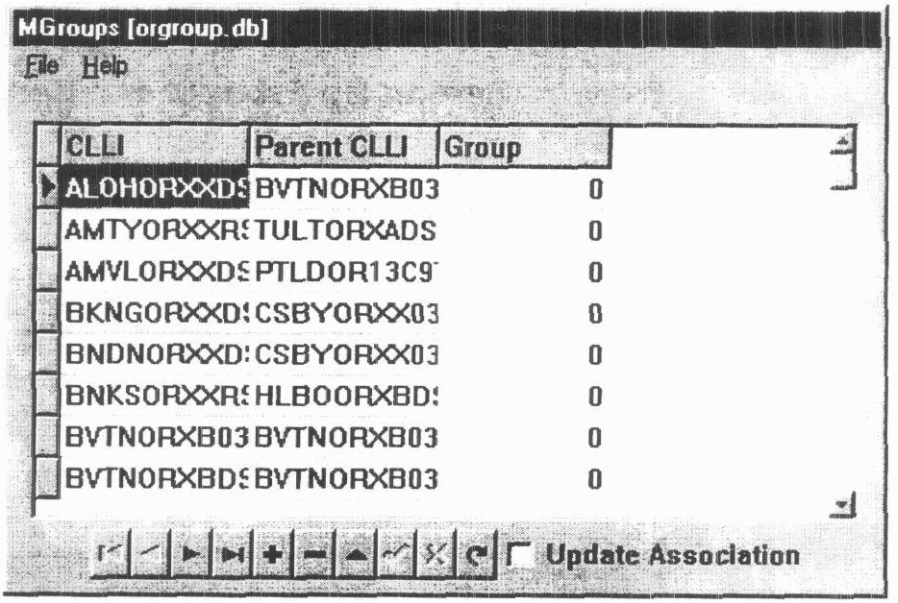
Adding Components to the Wizard

You can add components to the mapping wizard by selecting **Table/Product Mapping/Wizard Components**.

For information about the query function, see "Viewing and Searching Data Tables" on page 28. For information about importing and exporting data, see "Importing Data Files" on page 43 and "Exporting Specific Input Data Tables" on page 47.

Editing the Mapping Groups

You can edit the mapping groups that let you create reports for groups of wire centers.



To edit a mapping group:

1. Select **Table/Product Mapping/Mapping Group**.

   - The system displays the Mgroups windows.

2. Edit the groups the same way you edit other data tables.

   - For more information, see "Editing Data Tables" on page 40.

   - You can group wire centers by giving them the same group number.

   - You can load CLLI codes into the table by selecting **File/Load CLLIs.**

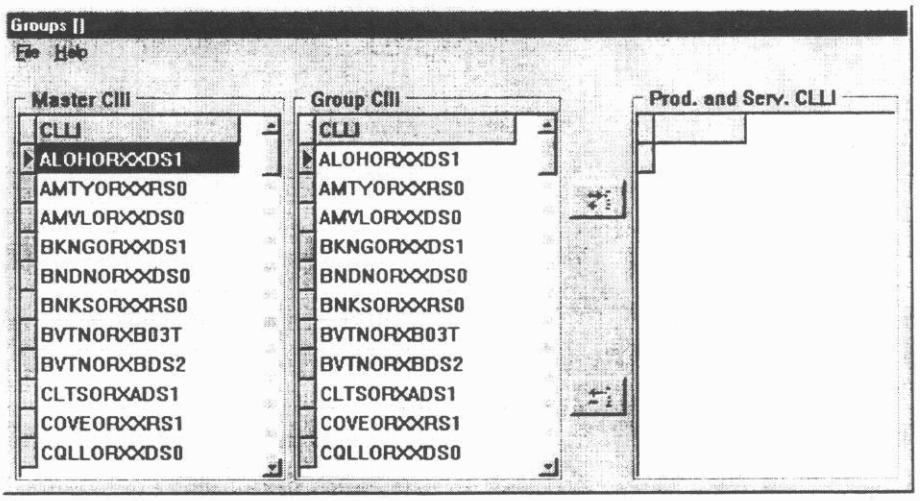
If the system displays a floating decimal error when you edit data, you should check your input. The error means that the value you entered is not a floating-point number.

Setting Up Product and Service Groups

You can set up groups for products and services. For example, you can set up how CLLI codes map for Extended Area Service (EAS). Each CLLI code can have a unique group of CLLI codes mapped to it for EAS.

To set up product and service groups:

1. Select **Table/Product Mapping/Product and Service Group**.
   - The system displays the Groups window.



2. Select **File/New**.
   - The system displays the Save As window.
3. Enter a file name for the group file you want to create.
   - Use .db as the file extension.
4. Click **SAVE**.
5. Select the CLLI code from the Master CLLI list that you want to create the group for.
6. Select a CLLI from the Group CLLI list that you want to add to the group.
7. Click the  button add the CLLI code to the group.

8. Repeat steps 6 and 7 for each CLLI that you want to group with the master CLLI.
9. Repeat steps 5 through 8 for each CLLI code from the Master CLLI column that you want to create a group for.
    - Your changes are automatically saved in the file you created in step 3.
    - You can remove a CLLI code from a group by selecting the code from the Prod. and Serv. CLLI column and clicking the  button.

You can now use this group file to generate reports for products and services.

To use the group file:

1. Select **Table/Product Mapping/Product and Services**.
    - The Product and Services Mapping window display.
2. Click the  button.
    - The system displays the Open window.
3. Select the group file you created.
4. Click **OPEN**.
    - When you run the model or print reports, the system processes the data by group. For more information see "Running the Model" on page 55 or "Generating Reports" on page 7.

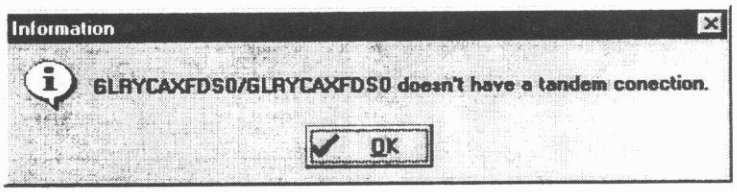
Running the Model

After you change any of the inputs or mappings, you must run the model. The model then uses the new data in the calculations.

To run the model:

1. Select **Report/Run**.
    - The system begins processing all of the data and displays a progress bar. The processing may take one to two hours, depending on the speed of your computer.

- If you see messages similar to either of the ones below, you must click **OK** or press **Enter** to continue processing.
- The message below shows that the switch is either a Class 4/5 (end office/tandem) switch or it is in the same building as the tandem switch and therefore has no trunk between it and the tandem.



Recreating Index Files

When you move ICM-related files from one location to another, you must be sure to include all of the files. You know that you are missing files if you get the following error when you try to run the model: "Index is out of date."

You can use the ICM pack/reindex function to recreate the missing index files.

To recreate index files:

1. Select **Utilities/Pack/Reindex Table**.
   - The Open window displays.
2. Select the table file that caused the error message: "Index is out of date."
3. Click **OPEN**.
   - The system displays a message letting you know that the process is complete.

Appendix A: Programming Language Alpha (PL-A)

PL-A General Considerations and Requirements

ICM includes a resident compiler (Programming Language Alpha or PL-A) that enables you to control calculations at critical points in the program. The compiler generates machine code and stores it for future use. This method of solving your mapping equations facilitates a more rapid execution time for generating the report tables.

The following information provides guidelines for using PL-A and implementing your mapping formulas.

Operators

All standard math operators are supported as well as logical operators.

Math operators for expressions

* , / , + . - , and ^ : Multiplication, division, addition, subtraction and power.

A*B, A/B, A+B, A-B, A^B.

Logical Operators (used in IF statements)

=, > , < , > = , < = , <> : Equality, greater than, less than, greater than or equal to, less than or equal to, not equal to.

A=B, A>B, A<B, A>=B, A<=B, A<>B

Assignment of values to variables

Assignment of values to variables is accomplished with the := operator

```
A := 5;
Z := 3;
B := A+Z;
C := B + (2^3 * A) + (A*B)/(Z*A);
```

Syntax format and requirements for writing formulas

- Each line of code must end with a semi-colon.
- Spaces are ignored.
- The compiler is case insensitive.
- Braces are used to enclose comments {this is a comment}.

You may use any name as a variable name except for the keywords used in flow control statement.

Flow Control Statements

For(Index, Start value, End value {,increment})

The function FOR is used for loops. All the arguments for the FOR function are numeric. A FOR function MUST have an END.FOR at the end of the loop. All Statements between the FOR an END.FOR will be executed for each iteration. The increment, if specified, must be positive.

Examples:

```
N := 2;
FOR (I, 3, 20);
  N := N * I;
END.FOR;
```

```
A := 2;
FOR (I,3,20,1);
  A := A * I + 2;
    IF (A > I);
      I := I + 1;
    END.IF;
END.FOR;
```

IF (Condition)

The IF function is used to compare values. It MUST have an END.IF statement. The IF statement may also have an optional ELSE statement. The logical operators are used for comparisons.

Examples:

```
If (a = x);      IF (r = 4);      IF (A >= B);   IF(A <> B);
  R  := x;          w := 3 / 4;          C := A;    A := B;
END.IF;          ELSE;            END.IF;        END.IF;
                 W :=  1 / 4;
                 END.IF;
```

Math functions

The following math functions can be used in your expressions.

ABS (Number)

Returns the absolute value of a number.

Examples:

```
ABS(-1) = 1
ABS(2.5) = 2.5
ABS(-6.1) = 6.1

x := ABS(x);
```

LN (Number)

Returns the natural log of a number. The natural log of X is the power to which *e* is raised to obtain X. The number *e* is approximately 2.71828.

Examples:

```
X := LN(X);

LN(2.71828) = 1
```

FRAC (Number)

Returns the fractional part of a number.

Example:

```
Xf := FRAC(3.625);     { Xf  equals .625}
```

INT (Number)

Returns the integer part of a number.

Example:

```
XInt := INT(2.5);       {Xint equals 2}
```

EXP (Number)

Returns a number that is *e* raised to the power of the number passed to the function.

Example:

```
Y := ln(5);
X := EXP(Y);
```

ROUND (Number)

Used to round a number to the nearest integer. Rounds off to the nearest whole number.

Examples:

```
Xround := ROUND(x);
ROUND (3.75) = 4
ROUND (2.5) = 3
ROUND (5.2) = 5
```

SIN (Number)

Returns the sine of a number expressed in Radians.

Example:

```
PI := 355/113;
X := SIN(PI/4);
```

SQR (Number)

Returns the square of a number.

Example:

```
X := 4;
Xx := SQR(X);
```

SQRT (Number)

Returns the square root of a number.

Example:

```
Xx := 4;
X := SQRT(Xx);
```

Mapping Examples

Call Forwarding Example (Component)

```
{THIS SECTION DOCUMENTS VARIABLE USAGE}
{ F0016 - Call Forwarding Variable CTX }
{ AD221200 - Maintenance and Support }
{ AC221200 - Property Tax }
{ AB221200 - Composite Tax }
{ AA221200 - Depreciation and Return }
{ Cost - Return value UNE Cost }
{ Invest - Total Investment }
{ DepRet - Return value Depreciation and Return }
{ CompTax - Return value Composite Tax }
{ PropTax - Return value Property Tax }
{ MntSprt - Return value Maintenance and Support }
{ Market - Return value Marketing }
{ BCDir - Return value Billing and Collection, Directory }

{ THIS SECTION DEFINES VARIABLES FOR
COMPILER}
Invest  := 0.0;
DepRet  := 0.0;
CompTax := 0.0;
PropTax := 0.0;
MntSprt := 0.0;
Market := 0.0;
BCDir := 0.0;

{THIS SECTION DEFINES EXPRESSIONS FOR
DETERMINING RETURN RESULT}
{ Call Forwarding Variable CTX}
Invest  := Invest  + .F0016;
DepRet  := DepRet  + .F0016 * .AA221200;
CompTax := CompTax + .F0016 * .AB221200;
PropTax := PropTax + .F0016 * .AC221200;
MntSprt := MntSprt + .F0016 * .AD221200;


Cost :=(DepRet + CompTax + PropTax + MntSprt)/12;
```

Explanation: The documentation section explains what each variable stands for, the "define variables for compiler" section defines and initializes the variables for use by the

compiler, and the expressions section contains the actual calculations to obtain the result passed on to the program for determining the cost. In this case the cost is equal to the sum of 4 different calculations divided by 12.

Three Way Calling Example (Component)

```
{THIS SECTION DOCUMENTS VARIABLE USAGE}
{ AC221200 - Property Tax }
{ AB221200 - Composite Tax }
{ AA221200 - Depreciation and Return }
{ Cost - Return value UNE Cost }
{ Invest - Total Investment }
{ DepRet - Return value Depreciation and Return }
{ CompTax - Return value Composite Tax }
{ PropTax - Return value Property Tax }
{ MntSprt - Return value Maintenance and Support }
{ Market - Return value Marketing }
{ BCDir - Return value Billing and Collection, Directory }

{THIS SECTION DEFINES VARIABLES FOR
COMPILER}
Invest  := 0.0;
DepRet  := 0.0;
CompTax := 0.0;
PropTax := 0.0;
MntSprt := 0.0;
Market := 0.0;
BCDir := 0.0;


{THIS SECTION DEFINES EXPRESSIONS FOR
DETERMINING RETURN RESULT}
{ Three Way Calling}
Invest  := Invest  + .F0001;
DepRet  := DepRet  + .F0001 * .AA221200;
CompTax := CompTax + .F0001 * .AB221200;
PropTax := PropTax + .F0001 * .AC221200;
MntSprt := MntSprt + .F0001 * .AD221200;


Cost :=(DepRet + CompTax + PropTax + MntSprt)/12;
```

Explanation: The documentation section explains what each variable stands for, the "define variables for compiler" section defines and initializes the variables for use by the compiler, and the expressions section contains the actual calculations to obtain the result passed on to the program for determining the cost. In this case the cost is equal to

the sum of 4 different calculations divided by 12. You may note that some variables are preceded by a period (.). These variables are automatically inserted by the wizard and are a special kind of variable that performs table lookups to obtain their value. A look at the features table will show that the only difference between this example and the previous example is in the expression section where the values for features F0001 and F0016 are extracted and implemented in the calculations.

Appendix B: Using SAS to Develop Final Service Costs

Installing SAS

Not applicable in states where results for products and services are not provided.

To generate the final service costs, you must send the ICM BNF output report to SAS. The installation instructions below assume that you already have SAS installed on your computer. You must install two additional files as described below to use SAS with ICM.

To install SAS:

1. Copy the files `instal~2.pif` and `icmrep~2.lnk` to your `c:\windows\desktop` directory.
   - This assumes that c: is your hard drive. You will see two new icons appear on your desktop.
2. Create a directory called `icm_sas`.
3. Right-click the **Install ICM Report Module** icon.
4. Click **Properties**.
5. Click the Program tab.
6. Make sure that the `cmdline` line has `d:\icmsas.exe -d`, where d:\ is the path of your self extracting zip file.

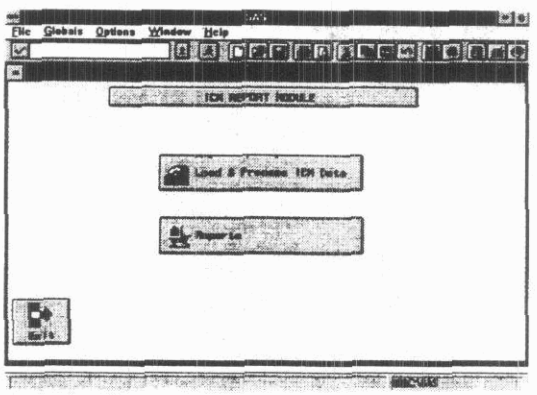
7. Make sure that the working line has `c:\icm_sas`, where c:\ is the path of where you put your `icm_sas` directory in step 2.
8. Click **OK** to exit Properties.
9. Double-click the install icon.
   - The ICM SAS files extract to your `icm_sas` directory.
10. To run ICM SAS, click the **ICM Report Module** icon.

Exporting the BNF Output Report to SAS

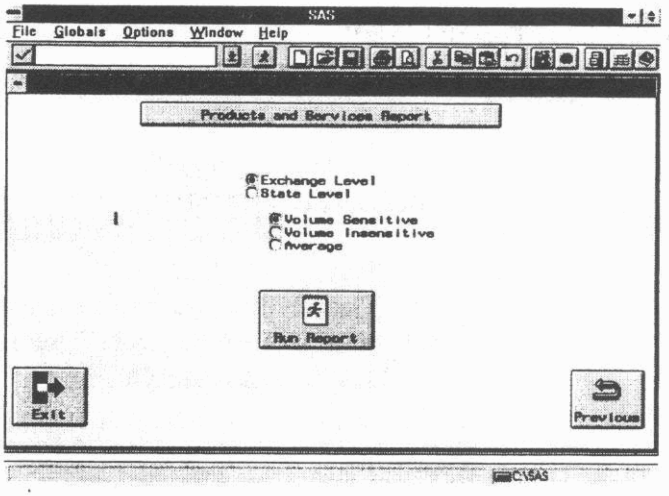
The BNF output report is used in external studies to develop service costs. You can export the report to SAS to develop final service costs.

To export the BNF output report to SAS:

1. In ICM, select **Utilities/Export to SAS.**
   - An output file, named `invest.text` is created and placed in the ICM directory. This assumes that you installed ICM in the default directory. If you did not, you should edit the `autoexec.sas` file and change the line that reads %let icm pth + \icm to reflect the directory where you installed ICM.
2. Start ICM SAS (#10 - Windows 95).



3. Click **Load and Process ICM Data**.
   - The ICM SAS program converts the text file to data sets which will be used to create reports.
4. Click **Reports** to see the report options.



5. Select whether you would like to run the reports at the Exchange or State Level.
6. Click **Run Report** to print reports.



Release 4.1

System Manual

(Book I of II)

GTE Service Corporation

Copyright GTE 1997 – 2000. All Rights Reserved

This is a Blank Page

ICM System Manual Package

Contents

System Manual Database Tables Book I of II

System Manual Codes Book II of II

This is a Blank Page

Table of Contents

ICM SYSTEM MANUAL PACKAGE ..... **A**

SYSTEM MANUAL DATABASE TABLES ..... BOOK I OF II

PREFACE **1**

ICM TABLES **2**

ACTCMPNT 2
ACTH2H 2
ACTLOOP 2
ACTMBNF 2
ACTPS 3
ACTSS7H 3
ACTSS7R 3
ACTSWITCH 3
ACTUNE 3
ACTUSF 3
AMVDEF 4
ARCFILE 4
AVLPLN 4
BNFDESC 4
BNFGROUP 5
BNFHDR 5
BNFMAP 5
BNFSTATE 5
CMPNT 5
DCL 5
DEMAND 5
DEMAND STYLE 6
EFI 6
EXPBASE 6
EXPCCWORK SHEET 7
EXPENSE 7
EXPJOINT SHARED 8
EXPRSLT 8
FEATURE 9
GROUP 9
H2H 9
H2R 9
HRARC 9
IARC 10
INTERRING 10
INVENTORY 10
IOFSET 10
LABR 10
MAPVAR 11
MATERIAL 11
MONITOR 11
NBNF 11
NODES 11

PLNTLIFE 12
PROPTX 12
PSGROUP 13
PSHDR 13
PSMAP 13
PSSTATE 13
ROEXPNSE 13
ROOSP 13
ROUSF 15
SSP 15
STP 15
STP LOCATION 16
SS7 INVESTMENT 16
SS7 NETWORK PARAMETERS 16
SS7 LOADING FACTOR 16
STATES 17
SWINV 17
SWITCH 17
TANDEMS 17
UNEGROUP 17
UNEHDR 17
UNEMAP 17
UNESTATE 17
USAGEP 18
USETABLE 18
USFHDR 18
USFMAP 18
USFRATES 18
USFSTATE 19
USGRATE 19
WIZARD 19

Preface

The Integrated Cost Model is written in Delphi, an object–oriented visual programming environment for rapid application development, using Object Pascal language and the Borland Database Engine. The *System Manual* is divided in two parts and in separate booklets. Part I comprises of ICM Tables. The tables have four columns: Table Name, Column Name, Type (i.e., data type, whether the data is a text, integer or True/False option), and Table's Purpose. The last column is added in ICM 4.1 to help the user understand the purpose of each table. Part II of the System Manual contains Program modules (pas.files).

A separate document, *ICM Model Methodology* provides a description of data inputs, model assumptions, algorithms, and model outputs. For information regarding how to install ICM, run reports, view data, and perform sensitivity analyses, see the *ICM User Guide*.

ICM Tables

This section includes a description and purpose of each ICM table. The state abbreviations that often appear as the first two letters of ICM table names are omitted. Table column headings and the type and length of text for the corresponding data are provided.

ACT tables are the output tables. An ACT table may be cost tables, or it may be an investment table, sometimes it may be a unit table, or sometimes it could be all the three –cost, investment and units - in the same table.

Table Name	Column Name	Type	Table's Purpose
ACTCMPNT	CLLI	Text	Contains Investment Components, which are used by mapping module to derive the basic network functions.
	Account	Text	
	Component	Text	
	Category	Text	
	Total Inv	Number(Double)	
	VS Inv	Number(Double)	
	VIS Inv	Number(Double)	
ACTH2H	CLLI	Text	Host to Host Interoffice Transport Investment outputs and units by Component for each Wire Center.
	Account	Text	
	Component	Text	
	Category	Text	
	Total Inv	Number(Double)	
	VS Inv	Number(Double)	
	VIS Inv	Number(Double)	
ACTLOOP	CLLI	Text	Loop Investment by Component for each Wire Center.
	Account	Text	
	Component	Text	
	Category	Text	
	Total Inv	Number(Double)	
	VS Inv	Number(Double)	
	VIS Inv	Number(Double)	
ACTMBNF	CLLI	Text	Basic Network Functions (BNF) level investment results by CLLI code.
	Item	Text	
	Weight	Number(Double)	
	Total Inv	Bytes	
	VS Inv	Bytes	
	VIS Inv	Bytes	

Table Name	Column Name	Type	Table's Purpose
ACTPS	CLLI	Text	Product and Services level investment results by CLLI code.
	Item	Text	
	Weight	Number(Double)	
	Total Inv	Bytes	
	VS Inv	Bytes	
	VIS Inv	Bytes	
ACTSS7H	CLLI	Text	SS7 component investment and cost results by host/standalone CLLI code.
	Account	Text	
	Component	Text	
	Category	Text	
	Total Investment	Number(Double)	
	VS Inv	Number(Double)	
	VIS Inv	Number(Double)	
ACTSS7R	CLLI	Text	SS7 component investment and cost results by remote CLLI code. Since all SS7 processing is performed by the host switch, the host values are simply populated for each of its remotes.
	Account	Text	
	Component	Text	
	Category	Text	
	Total Investment	Number(Double)	
	VS Inv	Number(Double)	
	VIS Inv	Number(Double)	
ACTSWITCH	CLLI	Text	Switch investment output and units by Component for each wire center.
	Account	Text	
	Component	Text	
	Category	Text	
	Total Inv	Number(Double)	
	VS Inv	Number(Double)	
	VIS Inv	Number(Double)	
ACTUNE	CLLI	Text	Unbundled Network Elements (UNE) level investment results by CLLI code.
	Item	Text	
	Weight	Number(Double)	
	Total Inv	Bytes	
	VS Inv	Bytes	
	VIS Inv	Bytes	
ACTUSF	CLLI	Text	Universal Service Fund (USF) level investment results by CLLI code.
	Item	Text	
	Weight	Number(Double)	
	Total Inv	Byte	
	VS Inv	Byte	
	VIS Inv	Byte	

Table Name	Column Name	Type	Table's Purpose
AMVDEF	Field Number	Integer	Definition of additional mapping variable. Contains definition of user defined variables for mapping.
	Field Name	Text	
	Mapping Variable	Text	
	Description	Text	
ARCFILE	CLLI	Text	Geographic and demand parameter that defines the Arcs in the local loop network. Data is given for Distribution (Arc type = 4), Non-Core feeder/distribution (Arc type = 3), Core-feeder (Arc type = 2), Mainfeeder (Arc type = 1) and shared Inter Office Transport Route (Arc type = 0).
	Arc Type	Integer	
	From X	Number(Double)	
	From Y	Number(Double)	
	To X	Number(Double)	
	To Y	Number(Double)	
	Node	Integer	
	X	Number(Double)	
	Y	Number(Double)	
	Length	Number(Double)	
	Dist to Origin	Number(Double)	
	Demand	Integer	
	Looplencode	Integer	
	Res Lines	Integer	
	Bus Lines	Integer	
	PerAerial	Number(Double)	
	PerBuried	Number(Double)	
	PerUndgnd	Number(Double)	
	BedRock	Number(Double)	
	WaterTable	Number(Double)	
	Elevation	Integer	
	Highway	Integer	
	Canplow	Integer	
	Structure	Integer	
	IOF	Number(Double)	
	HRF	Number(Double)	
	IRF	Integer	
	OSP Type	Number(Double)	
AVLPLN	CLLI	Text	Table used to generate the Average Loop Length Report.
	PairFeet	Integer	
BNFDESC	BNF	Text	Definition codes of Basic Network Functions (BNF).
	Description	Text	

Table Name	Column Name	Type	Table's Purpose
BNFGROUP	CLLI	Text	Investment and weighting factors by BNF code for each group of offices.
	Item	Text	
	Weight	Number(Double)	
	Total Inv	Bytes	
	VS Inv	Bytes	
	VIS Inv	Bytes	
BNFHDR	Column	Integer	Definition of column names within the BNF table.
	Label	Text	
	Size	Integer	
	Format	Text	
	Variable	Text	
BNFMAP	Order	Integer	Contains the definition of each BNF for the state.
	BNF Code	Text	
	Indent	Small/Integer	
	Label	Yes/No	
	Format	Text	
	Description	Text	
	MapCode	Memo	
	ExeCode	Binary	
BNFSTATE	CLLI	Text	Investment and weighting factors by BNF code for the state.
	Item	Text	
	Weight	Number(Double)	
	Total Inv	Bytes	
	VS Inv	Bytes	
	VIS Inv	Bytes	
CMPNT	Component	Text	Definition of all Component codes calculated within the ICM modules and used in the mapping equations.
	Category	Text	
	Account	Text	
	Description	Text	
DCL	CLLI	Text	Geographic coordinates of each cluster modeled in the local loop for each wire center.
	X	Integer	
	Y	Integer	
	Latitude	Number(Double)	
	Longitude	Number(Double)	
DEMAND	CLLI	Text	Provides line demand and geographic information by CLLI code and by demand point for use in loop cost calculations.
	X	Number(Double)	
	Y	Number(Double)	
	Map Info	Text	
	Road Length	Number(Double)	
	BedRock	Number(Double)	

Table Name	Column Name	Type	Table's Purpose
	Water Table	Number(Double)	
	Res Lines	Number(Double)	
	Bus Lines	Number(Double)	
	Water Flag	True/False	
	Elevation Code	Number(Double)	
	Highway flag	True/False	
	X Cluster	Number(Double)	
	Y Cluster	Number(Double)	
DEMAND STYLE	Minval	Integer	Paremeters that define the layout design of each distribution template.
	Maxval	Integer	
	RCOEF1	Number(Double)	
	RCOEF2	Number(Double)	
	RCOEF3	Number(Double)	
	RCOEF4	Number(Double)	
	RCOEF5	Number(Double)	
	RCOEF6	Number(Double)	
	RCOEF7	Number(Double)	
	RCOEF8	Number(Double)	
	DCOEF1	Number(Double)	
	DCOEF2	Number(Double)	
	DCOEF3	Number(Double)	
	DCOEF4	Number(Double)	
	DCOEF5	Number(Double)	
	DCOEF6	Number(Double)	
	DCOEF7	Number(Double)	
	DCOEF8	Number(Double)	
	PLWFLG	Integer	
	STRFLG	Integer	
	STRHDT	Integer	
	AvgLenFrac	Number(Double)	
	DDT	Integer	
	ARC4DT	Integer	
EFI	Switch Type	Text	Engineering, Furnished and Installed factors, which account for the GTE labor to engineer and install switch. Factors are given by switch type and switch size
	Size	Number(Double)	
	EF-I	Number(Double)	
	EFI-EF	Number(Double)	
	IAF	Number(Double)	
	Power Inv	Number(Double)	
	Test Inv	Number(Double)	
	Loop Test	Number(Double)	
EXPBASE	Long Term Factor	Number(Double)	An input table which includes information related to the rate of return, the capital structure, the applicable tax rates and revenue, used within ICM.
	Long Term Rate	Number(Double)	
	Short Term Factor	Number(Double)	

Table Name	Column Name	Type	Table's Purpose
	Short Term Rate	Number(Double)	
	Common Fact	Number(Double)	
	Common Rate	Number(Double)	
	PRFD Fact	Number(Double)	
	PRFD Rate	Number(Double)	
	Rate Return	Number(Double)	
	State Rate	Number(Double)	
	Federal Rate	Number(Double)	
	Property Tax	Number(Double)	
	Total Revenue	Number(Double)	
	Date Year	Number(Long)	
EXPCCWORK SHEET	Account	Text	This is an output report prepared within ICM used to compute common cost. Data used in the worksheet is derived from the EXPENSE.db file.
	Category	Text	
	Amount	Number(Double)	
	Adjust1	Number(Double)	
	Adjust2	Number(Double)	
	Adjust3	Number(Double)	
	New Amount	Number(Double)	
	%BC	Number(Double)	
	%Consumer	Number(Double)	
	%Business	Number(Double)	
	%CXR	Number(Double)	
	%Common	Number(Double)	
	$BC	Number(Double)	
	$Consumer	Number(Double)	
	$Business	Number(Double)	
	$CXR	Number(Double)	
	$Common	Number(Double)	
	SubTotal	Number(Double)	
	ICM Based Factors	Number(Double)	
	Total	Number(Double)	
EXPENSE	Account	Text	An input table that includes ARMIS expense and investment data, applicable adjustments (i.e., Adjust1, Adjust2, Adjust3) and cost pool assignments to be used in ICM.
	Category	Text	
	Amount	Number(Double)	
	Adjust1	Number(Double)	
	Adjust2	Number(Double)	
	Adjust3	Number(Double)	
	Default Adjust3	Number	
	Cable	Number(Double)	
	FiAer	Number(Double)	
	CuAer	Number(Double)	
	FiBur	Number(Double)	
	CuBur	Number(Double)	
	FiUnd	Number(Double)	
	CuUnd	Number(Double)	

Table Name	Column Name	Type	Table's Purpose
	Poles	Number(Double)	
	Conduit	Number(Double)	
	Transmission	Number(Double)	
	Switch	Number(Double)	
	Access	Number(Double)	
	IOT	Number(Double)	
	Direct Other	Number(Double)	
	Bill_Pay	Number(Double)	
	Oper_Serv	Number(Double)	
	PubCom	Number(Double)	
	Consumer	Number(Double)	
	Business	Number(Double)	
	CXR	Number(Double)	
	Common	Number(Double)	
EXPJOINT SHARED	Account	Text	An output report from ICM, which shows the components of network services shared cost. Data used in the EXPJOINTSHARED report is derived from EXPENSE.db file.
	Category	Text	
	Total	Number(Double)	
	Cable	Number(Double)	
	Aerial Non-Metallic	Number(Double)	
	Aerial Metallic	Number(Double)	
	Buried Non-Metallic	Number(Double)	
	Buried Metallic	Number(Double)	
	Underground Non-Metallic	Number(Double)	
	Underground Metallic	Number(Double)	
	Poles	Number(Double)	
	Conduit	Number(Double)	
	Transmission	Number(Double)	
	Switch	Number(Double)	
	IOT	Number(Double)	
	Direct Other	Number(Double)	
	Access	Number(Double)	
	B and C	Number(Double)	
	Oper	Number(Double)	
	PUB COMM	Number(Double)	
	Consumer	Number(Double)	
	Business	Number(Double)	
	CXR	Number(Double)	
	Common	Number(Double)	
EXPRSLT	Account	Text	An output table from the expense module which contains the computed capital recovery factor, composite income tax factor, pool factor and property tax factor for related GTE Service Corporation accounts.
	Depreciation and Return	Number(Double)	

Table Name	Column Name	Type	Table's Purpose
	Composite Income Tax	Number(Double)	
	Maintenance and Support	Number(Double)	
	Property Tax	Number(Double)	
FEATURE	Feature	Text	Listing of feature names corresponding to ICM feature codes.
	Category	Text	
	Description	Text	
	Group	Text	
GROUP	CLLI	Text	If necessary, wire centers may be placed with a group that is smaller than the entire state. Thus results could be obtained for any subset of GTE offices within a state.
	Parent CLLI	Text	
	Group	Integer	
H2H	ECLLI	Text	Host-to-Host Interoffice Transport demand data by CLLI code, such as switch DS-1 ports and special access lines by type.
	Hub CLLI	Text	
	GCLLI	Text	
	Ring	Number(Long)	
	RemoteSW	Number(Long)	
	SWPorts	Number(Long)	
	DS0SAL	Number(Long)	
	DS1SAL	Number(Long)	
	DS3SAL	Number(Long)	
H2R	HCLLI	Text	Host-to-Remote Interoffice Transport demand data by CLLI code, such as switch DS-1 ports and special access lines by type.
	RCLLI	Text	
	SWPorts	Number(Long)	
	DS0SAL	Number(Long)	
	DS1SAL	Number(Long)	
	DS3SAL	Number(Long)	
HRARC	CLLI	Text	Geographic parameters that define the route linking host and remote offices.
	Arc Type	Integer	
	From X	Number(Double)	
	From Y	Number(Double)	
	To X	Number(Double)	
	To Y	Number(Double)	
	RCLLI	Text	
	Node	Integer	
	X	Number(Double)	
	Y	Number(Double)	
	Distance	Integer	
	Ring Dist	Number(Double)	

Table Name	Column Name	Type	Table's Purpose
IARC	TCLLI	Text	Geographic parameters that define each sonnet ring in GTE Interoffice Transport network. Data is given for distances between each pair of offices on a ring. If another operating company is involved in the route, then the distance on their facilities is also given.
	Ring	Integer	
	RingPos	Integer	
	CLLIA	Text	
	CLLIA Latitude	Number(Double)	
	CLLIA Longitude	Number(Double)	
	CLLIB	Text	
	CLLIBLatitude	Number(Double)	
	CLLIB Longitude	Number(Double)	
	GCLLI Flag	True/False	
	InterRing Flag	True/False	
	Distance	Number(Double)	
	OSConn	Number(Double)	
INTERRING	ICLLI	Text	InterRing Interoffice Transport demand data by CLLI switch DS-1 ports.
	ICLLI Substitute	Text	
	IHubCLLI	Text	
	IHubCLLI Substitution	Text	
	IRing	Integer	
	ISWPorts	Integer	
INVENTORY	CLLI	Text	Inventory file containing a list of materials for outside pland such as cables, poles, conduit, etc.
	OSP Type	Text	
	Component	Text	
	Size	Integer	
	X	Integer	
	Y	Integer	
	Units	Integer	
IOFSET	Admin Fill	Number(Double)	Setting for Interoffice facility parameters such as administration fill factor.
	IntraRing Factor	Number(Double)	
	Aerial Span	Number(Double)	
	Buried Span	Number(Double)	
	Air2Route	Number(Double)	
LABR	Labor Type	Text	Engineering, Installation and labor costs by equipment type and size.
	Description	Text	
	Tier A	Number(Double)	

Table Name	Column Name	Type	Table's Purpose
	Tier B	Number(Double)	
MAPVAR	See AMVDEF, mapping variables field, for field definition of this table.		Contains user defined variables for mapping.
MATERIAL	Material Size Size Description Unit Price Minor Material	Text Number Text Number True/False	Material hardware prices by equipment type and size loop and interoffice transport cost
MONITOR	Log Number Module Level Caller Text	Integer Text Integer Text Text	Diagnostic log file for ICM session.
NBNF	Item Description Average VS VIS	Text Text Number(Double) Number(Double) Number(Double)	An input table that includes the Activity Base Cost (ABC) /Factors associated with service assurance/or repair and maintenance, sales marketing and advertising, billing and collection and directory cost by average cost, volume sensitive (VS) and volume insensitive (VIS) components.
NODES	CLLI Host CLLI Switch Type Class HML Latitude Longitude Total Lines Non Switched Lines Demand Points Lines Terms	Text Text Text Number(Long) Text Number(Double) Number(Double) Number(Double) Number(Double) Integer Number(Double)	Line demand and geographic data by CLLI code for the state. The table also includes switch type and percentage of cable as aerial, buried and underground.

Table Name	Column Name	Type	Table's Purpose
	BRI	Number(Double)	
	PRI	Number(Double)	
	PBX	Number(Double)	
	KEY	Number(Double)	
	CENTRANET	Number(Double)	
	COIN	Number(Double)	
	B1	Number(Double)	
	Road Feet	Number(Double)	
	RoadFeetRatio	Number(Double)	
	BedRock	Number(Double)	
	WaterTable	Number(Double)	
	Cluster Count 12Kft	Integer	
	Distance Adj	Number(Double)	
	WidthDemandPt	Integer	
	HeightDemand Pt	Integer	
	DU	Number(Double)	
	FU	Number(Double)	
	F Aerial Pct	Number(Double)	
	F Buried Pct	Number(Double)	
	F Undgnd Pct	Number(Double)	
	C Aerial Pct	Number(Double)	
	C Buried Pct	Number(Double)	
	C Undgnd Pct	Number(Double)	
	Tier B	Yes/No	
PLNTLIFE	Account	Text	An input table which includes information related to economic depreciable life and salvage, book life and salvage, and Modified Accelerated Cost Recovery (MACRS) life for all investment accounts. This information is used in ICM to compute depreciation expense capital recovery and composite income tax rates.
	Economic Life	Number(Double)	
	Econ Salvage	Number(Double)	
	Book Life	Number(Double)	
	Book Salvage	Number(Double)	
	Macrs Years	Number(Double)	
PROPTX	Account	Text	An input table which includes the property tax rate applicable to general support investments.
	Property Tax Flag	Yes/No	
	Property Tax	Number(Double)	

Table Name	Column Name	Type	Table's Purpose
PSGROUP	CLLI	Text	Grouping of CLLI codes within a state. The table reports retail service cost for a set of offices smaller than the entire state. It can only be viewed through the Maptext utility.
	Item	Text	
	Weight	Number(Double)	
	Total Inv	Bytes	
	VS Inv	Bytes	
	VIS Inv	Bytes	
PSHDR	Column	Integer	Investment and weighing factors by Product and Service (PS) for each group of offices.
	Label	Text	
	Size	Integer	
	Format	Text	
	Variable	Text	
PSMAP	Sequence	Integer	Contains the definition of each Product and Service (PS) for the state..
	Indent	Small Integer	
	Label	Yes/No	
	Format	Text	
	Description	Text	
	PS Table	Text	
	MapCode	Memo	
	ExeCode	Binary	
PSSTATE	CLLI	Text	Investment and weighting factors by PS code for the state.
	Item	Text	
	Weight	Number(Double)	
	Total Inv	Bytes	
	VS Inv	Bytes	
	VIS Inv	Bytes	
ROEXPNSE	Life	Text	User option expense table that allows user to change the option from wholesale to retail and from shared to non-shared.
	Market	Text	
	Inflation	Number(Double)	
	Productivity	Number(Double)	
	Horizon	Number(Double)	
	Shared	Number(Double)	
	Common Cost	True/False	
ROOSP	ED Feeder	Number(Double)	User option Outside Plant Table that allows the user to adjust the values of items such as drop length and engineering factor to conduct sensitivity study or make the values state specific.
	EDDistribution	Number(Double)	
	ED Adm Fill	Number(Double)	
	SFFPoles	Number(Double)	
	SFA	Number(Double)	
	SFB	Number(Double)	
	SFC	Number(Double)	
	SDFPoles	Number(Double)	

Table Name	Column Name	Type	Table's Purpose
	SDA	Number(Double)	
	SDB	Number(Double)	
	SDC	Number(Double)	
	DPWIRE	Number(Long)	
	Pole Spacing	Number(Double)	
	Manhole Spacing	Number(Double)	
	Pull Box Spacing	Number(Double)	
	XCONN	Number(Double)	
	Per Hand	Number(Double)	
	Per Concrete	Number(Double)	
	Per GuyWire	Number(Double)	
	SPU	Number(Double)	
	STU	Number(Double)	
	SCU	Number(Double)	
	Horizon	Number(Double)	
	Growth	Number(Double)	
	Fill Dist	Number(Double)	
	Fill Feeder	Number(Double)	
	UseFill Factor	Yes/No	
	LoopLength	Small Integer	
	AerCuSpan	Number(Double)	
	BurCuSpan	Number(Double)	
	AerFiSpan	Number(Double)	
	BurFiSpan	Number(Double)	
	LnFt Manhole	Number(Double)	
	LnFtPullbox	Number(Double)	
	Well Pointdays	Number(Double)	
	PerBoring	Number(Double)	
	XCSize1	Integer	
	XCSize2	Integer	
	Fdrop	Number(Double)	
	PerTrench	Number(Double)	
	FeedLineUnitW	True/False	
	DistDropMax	Number(Double)	
	DistDropMin	Number(Double)	
	DistUnitLRes	Number(Double)	
	DistUnitLBus	Number(Double)	
	DistTermRes	Number(Double)	
	DistTermBus	Number(Double)	
	DistThreBus1	Number(Double)	
	DistThreBus2	Number(Double)	
	DistDrpBus1	Number(Double)	
	DistDrpBus2	Number(Double)	
	DistThreRes1	Number(Double)	
	DistDrpRes1	Number(Double)	
	MinXCDist	Integer	

Table Name	Column Name	Type	Table's Purpose
ROUSF	BM Rate	Number(Double)	The table contains runtime options for USF costs and contains benchmark rates that are utilized to calculate the size of the USF fund estimates. The costs are compared to the cost benchmark and applied against units to determine USF support.
	Res Rate	Number(Double)	
	Res Offset	Number(Double)	
	Bus Rate	Number(Double)	
	Bus Offset	Number(Double)	
SSP	SSP_CLLI	Text	Service Switching Point (SSP) provides CLLI codes of the host/stand-alone offices in the state, along with the CLLI codes of STPs supporting the SSPs. As part of a forward-looking model, each host/stand-alone office is assumed to be SSP-equipped.
	Local_STP1	Text	
	Local_STP2	Text	
	Alinks	Integer	
	A1 Expense	Number (Double)	
	A2 Expense	Number (Double)	
STP	STP_CLLI	Text	Signal Transfer Point (STP) provides general data about STPs, such as geographic coordinates, port quantities, and expenses for links to other STPs.
	Wcenter_ CLLI	Text	
	STATE	Text	
	STP_TYPE	Text	
	STP_ LATITUDE	Number (Double)	
	STP_ LONG - ITUDE	Number (Double)	
	Ports_InSvc	Integer	
	PortCap	Integer	
	REGIONAL_STP1	Text	
	REGIONAL_STP2	Text	
	Clinks	Integer	
	Clink_ Distance	Number (Double)	
	Clink_ Expense	Number (Double)	
	Dlink_ Expense	Number (Double)	
	DB800Link_Exp	Number (Double)	
	LIDBLink_ Exp	Number (Double)	
	LNPLink_ Exp	Number (Double)	

Table Name	Column Name	Type	Table's Purpose
	AINLink_Exp	Number (Double)	
STP Location	REGIONAL_STP1	Text	Signal Transfer Point Location (STP Location) for each regional STP, indicates where each pair of service-specific SCPs is located.
	DB800_PriSCP1	Text	
	DB800_PriSCP2	Text	
	DB800_SecSCP1	Text	
	DB800_SecSCP2	Text	
	LIDB_PriSCP1	Text	
	LIDB_PriSCP2	Text	
	LIDB_SecSCP1	Text	
	LIDB_SecSCP2	Text	
	LNP_PriSCP1	Text	
	LNP_PriSCP2	Text	
	LNP_SecSCP1	Text	
	LNP_SecSCP2	Text	
	AIN_PriSCP1	Text	
	AIN_PriSCP2	Text	
	AIN_SecSCP1	Text	
	AIN_SecSCP2	Text	
SS7 INVESTMENT	Node_Type	Text	Provides SS7 hardware and software investment by network component, along with estimated labor hours.
	Item	Text	
	Technology	Text	
	Account	Number(Double)	
	Hardware	Number(Double)	
	Software	Number(Double)	
	Eng_Hrs	Number(Double)	
	Inst_Hrs	Number(Double)	
	Test_Hrs	Number(Double)	
SS7 Network Parameters	Code	Text	Provides SS7 network engineering and utilization data required for cost calculations.
	Description	Text	
	Value	Number(Double)	
SS7 Loading Factor	State	Text	Provides state-specific minor material loadings and labor rates to convert material investments. Calculated total investments then include all associated minor material and labor investments.
	Hw_Factor	Number(Double)	
	Sw_Factor	Number(Double)	
	Eng_Rate	Number(Double)	
	Inst_Rate	Number(Double)	

Table Name	Column Name	Type	Table's Purpose
	Test_Rate	Number(Double)	
STATES	ST_NAME	Text	Listing of all states for which ICM is being used. For this particular state, the path to the ICM database appears in the 'AliasPath' column. Text of States can be viewed by clicking "memo" within the MapCode column.
	AliasPath	Text	
	Active	True/False	
SWINV	CLLI	Text	Average and volume sensitive investments by switch feature, switch type & switch line size. The inputs to this table are the outputs from SCIS & COSTMOD switch investment model runs.
	Investment Type	Text	
	Category	Text	
	Average Investment	Number(Double)	
	VS Invest	Number(Double)	
SWITCH	Switch Type	Text	A listing of switch types used in the ICM Model.
	ID	Integer	
	Host/Remote	Text	
TANDEMS	TCLLI	Text	A listing of tandem offices serving GTE offices in the state. The table identifies the geographic coordinates and the company owning the tandems.
	COLOCLLI	Text	
	TND Company	Number(Double)	
UNEGROUP	CLLI	Text	Investment and weighting factors by UNE code for group of offices.
	Item	Text	
	Weight	Number(Double)	
	Total Inv	Bytes	
	VS Inv	Bytes	
	VIS Inv	Bytes	
UNEHDR	Column	Integer	Definition of column names within the UNE table.
	Label	Text	
	Size	Integer	
	Format	Text	
	Variable	Text	

Table Name	Column Name	Type	Table's Purpose
UNEMAP	Sequence	Integer	Contains the definition of each UNE for the state.
	Indent	Small Integer	
	Label	Yes/No	
	Format	Text	
	Description	Text	
	MapCode	Memo	
	ExeCode	Binary	
	Results	Bytes	
UNESTATE	CLLI	Text	Investment and weighting factors by UNE code forthe state.
	Item	Text	
	Weight	Number(Double)	
	Total Inv	Bytes	
	VS Inv	Bytes	
	VIS Inv	Bytes	
USAGEP	CallComp	Number(Double)	Switching usage parameters required for call set up cost calculations.
	SS7Setup	Number(Double)	
	Ring Time	Number(Double)	
	SS7 Call Setup	Number(Double)	
	Dial Time	Number(Double)	
USETABLE	Index	Number(Long)	Contains all the parameters for ICM to function.
	Show	Text	
	Lookup Table	Text	
	Filename	Text	
USFHDR	Column	Integer	Investment and weighing factors by USF for each group of offices..
	Label	Text	
	Size	Integer	
	Format	Text	
	Variable	Text	
USFMAP	Sequence	Text	Contains the definition of each USF for the state.
	Indent	Number(Double)	
	Label	Yes/No	
	Format	Text	
	Description	Text	
	MapCode	Memo	
	EXECode	Binary	
USFRATES	CLLI	Text	Residential and Business unit service rates by CLLI code.
	BM Rate	Text	
	Res Rate	Number(Double)	
	Res Offset	Number(Double)	
	Bus Rate	Number(Double)	

Table Name	Column Name	Type	Table's Purpose
	Bus Offset	Number(Double)	
USFSTATE	CLLI	Text	Investment and weighting factors by USF code for the state.
	Item	Text	
	Weight	Number(Double)	
	Total Inv	Bytes	
	VS Inv	Bytes	
	VIS Inv	Bytes	
USGRATE	Index	Number(Long)	Conversion factors between busy hours and rate period used to convert busy hour cost to cost by rate period.
	Description	Text	
	Day	Number(Double)	
	Evening	Number(Double)	
	Night	Number(Double)	
	Peak	Number(Double)	
	Off Peak	Number(Double)	
	24 Hour	Number(Double)	
WIZARD	Component	Text	Description of each BNF code used in ICM.
	Category	Text	
	Account	Text	
	Description	Text	



Integrated
Cost Model

Release 4.1

System Manual
(Book II of II)

GTE Service Corporation

Copyright © 1997 – 2000. All Rights Reserved

4 2

This is a Blank Page

ICM SYSTEM MANUAL PACKAGE

Contents

System Manual Database Tables Book I of II

System Manual Codes **Book II of II**

This is a Blank Page

Table of Contents

ICM SYSTEM MANUAL PACKAGE ..... **a**

Key Program Modules **1**

DEMAND.pas 2

DISTRIB.pas 22

EXPENSE.pas 34

INTEROFC.pas 116

IOFARC.pas 190

MAINDEMANDMAP.pas 212

NODES.pas 250

OUTSIDE PLANT (OSP).pas 263

OSPINVEST.pas 274

PLACE.pas 351

SKCLUSTER.pas 360

SPANNING.pas 369

SS7.pas 403

SWITCH.pas 551

This is a Blank Page

Key Program Modules

The program modules (Pascal or .pas files) are object oriented code that contain model functions and procedures. Copies of these files are found under the Source directory on the ICM CD-ROM. This section contains key modules. Algorithms that describe how the code models the telephone network are found in a separate document, *ICM Module Support Documentation.*

DEMAND.pas

```
unit Demand;
{

  This unit contains the routines to handle all of the Demand  information.
  The Demand table contain the information require the calculate the distribution
  and feeder parts of the loop.
}

interface

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
  Forms, Dialogs, Menus, DB, DBTables, Assoc, ExtCtrls, DBCtrls, Grids,
  DBGrids, StdCtrls, Tabs, Printers, Gauges, Invntry, GenPrnt, Import,
  Gen_Exp, ICMDM, OSPTypes, OSP, Osplabor, Ospmater, Monitr, Globalda,
  Interofc;

type
  Tf_Demand = class(TForm)
    MainMenu1: TMainMenu;
    File1: TMenuItem;
    Open1: TMenuItem;
    New1: TMenuItem;
    Exit1: TMenuItem;
    sd_Demand: TSaveDialog;
    Help1: TMenuItem;
    N1: TMenuItem;
    N3: TMenuItem;
    N2: TMenuItem;
    Print1: TMenuItem;
    PrintDialog1: TPrintDialog;
    Export1: TMenuItem;
    od_Demand: TOpenDialog;
    DB_Demand: TDBGrid;
    DBNavigator1: TDBNavigator;
```

```
37      cb_Demand: TCheckBox;
38      l_Copyright: TLabel;
39      ts_Demand: TTabSet;
40      Common1: TMenuItem;
41      Tab1: TMenuItem;
42      ds_Demand: TDataSource;
43      p_Progress: TPanel;
44      l_progress: TLabel;
45      Gauge1: TGauge;
46      Import1: TMenuItem;
47
48      procedure New1Click(Sender: TObject);
49      procedure Exit1Click(Sender: TObject);
50      procedure Open1Click(Sender: TObject);
51      procedure FormShow(Sender: TObject);
52      procedure FormResize(Sender: TObject);
53      procedure ts_DemandChange(Sender: TObject; NewTab: Integer;
54        var AllowChange: Boolean);
55      procedure FormClose(Sender: TObject; var Action: TCloseAction);
56      procedure Print1Click(Sender: TObject);
57      procedure Export1Click(Sender: TObject);
58      procedure FormActivate(Sender: TObject);
59      procedure Tab1Click(Sender: TObject);
60      procedure Common1Click(Sender: TObject);
61      procedure Import1Click(Sender: TObject);
62    private
63      { Private declarations }
64    public
65      { Public declarations }
66      Mode  : word;
67      procedure Entry;
68      procedure Process;
69      procedure OSP_StartUp;
70      procedure OSP_GetMemory;
71      Procedure OSP_FreeMemory;
72      procedure PrintDemand( Open : Boolean);
73    end;
74
75  var
```

```
76      f_Demand     : Tf_Demand;
77
78    implementation
79       uses NeMain;
80    {$R *.DFM}
81     Var
82      TabName      : String;
83
84    procedure Tf_Demand.Entry;
85     begin
86       ts_Demand.Enabled := True;
87       Mode := 2;
88       Show;
89     end;
90
91    procedure Tf_Demand.Process;
92     begin
93      if f_associate.GetFileName(100) = 'YES'
94        then begin
95               // initialize math coprocessor
96               ASM
97                fninit
98               end;
99               OSP_StartUp;
100              OSP_GetMemory;
101              // init actloop table
102              OSP_Table.Init;
103              OSP_Table.Make;
104              // init acth2h table
105              IOF_Table.Init;
106              IOF_Table.Make;
107              // Create a table for average loop length
108              AvgLpLength.Init;
109              AvgLpLength.Make;
110              // Create a table for osp inventory
111              PartInventory.Init;
112              PartInventory.Make;
113
114
```

```
115             f_OSPMaterial.Mat('OPEN',0);     // Open material table
116             f_OSPLabor.Labor('OPEN','XXXX'); // Open labor table
117             MaxPG := f_OSPMaterial.MaxSize ('LCBAR');
118             MaxXC := f_OSPMaterial.MaxSize ('XCONN');
119
120             WeAreWatching.LogEvent(1,'Process','Processing Demand');
121             ProcessID := 1;
122
123             LoadParameters;
124
125             AvgLpLength.Done;                 // Close table for average loop length
126             PartInventory.Done;               // Close table for osp inventory
127             f_OSPMaterial.Mat('CLOSE',0);     // Close material table
128             f_OSPLabor.Labor('CLOSE','XXXX'); // Close labor table
129             // close actloop table
130             OSP_Table.Done;
131             IOF_Table.Done;
132             OSP_FreeMemory;
133
134             // Close Open tables
135             DM_ICM.t_Arcs.Close;
136             DM_ICM.t_Nodes.Close;
137             DM_ICM.t_Demand.Close;
138             DM_ICM.t_DLC.Close;
139             Hide;
140
141             f_associate.SetFileName(100, 'NO');
142             f_associate.SetFileName(101, 'YES'); {Switch}
143
144           end;
145   end;
146
147   Procedure Tf_Demand.OSP_GetMemory;
148    begin
149      MaxObs := 0;
150      DM_ICM.t_Nodes.First;
151      while Not DM_ICM.t_Nodes.Eof do  // Loop through Node table one record a time
152        begin                          // Find the largest number of demand points
153          if DM_ICM.t_Nodes.FieldByName('Demand Points').AsInteger > MaxObs
```

```
154             then MaxObs := DM_ICM.t_Nodes.FieldByName('Demand Points').AsInteger;
155           DM_ICM.t_Nodes.Next;
156         end;
157       MaxObs := (MaxObs div 8 + 1) * 8; // Adjust Maximum observation to the next higher 256 boundary
158       DemandList := TList.Create;
159       DemandList.Capacity := MaxObs;
160
161       SAIList := TList.Create;
162       SAIList.Capacity := MaxObs;
163   end;
164
165   Procedure Tf_Demand.OSP_FreeMemory;
166    begin
167       DemandList.Clear;
168       DemandList.Free;
169
170       SAIList.Clear;
171       SAIList.Free;
172
173       FreeMem (DistStyle);
174   end;
175
176   procedure Tf_Demand.OSP_StartUp;
177
178   var
179     Index, I : Integer;    // Distribution Style Index number
180     Str      : String;
181
182   begin   //process
183     db_Demand.DataSource := Nil;
184
185   { Set data controls }
186     ts_Demand.TabIndex := 0;
187     ts_Demand.Enabled  := False;
188     ds_Demand.Enabled  := True;
189     Mode := 3;
190     Show;
191
192     // Set Run time Options for Outside plant run
```

```
193
194     With DM_ICM.t_Demand do // Open outside plant runtime option table
195      begin
196        if Active then Close;
197        DatabaseName := 'modata';
198        TableName := 'roosp.db';
199        TableType := ttDefault;
200        Exclusive := False;
201        IndexName := '';
202        IndexFieldNames := '';
203        Open;
204
205        With Asmp Do
206          begin
207            // Load outside plant runtime optiond
208            RdFtRatio           := 1.0;
209            RLnPUnit            := FieldByName('DistUnitLRes').AsFloat;
210            BLnPUnit            := FieldByName('DistUnitLBus').AsFloat;
211            RUnitPTerm          := FieldByName('DistTermRes').AsFloat;
212            BUnitPTerm          := FieldByName('DistTermBus').AsFloat;
213            MxDrpLn             := FieldByName('DistDropMax').AsInteger;
214            MnDrpLn             := FieldByName('DistDropMin').AsInteger;
215            BDSth1              := FieldByName('DistThreBus1').AsInteger;
216            BDSpr1              := FieldByName('DistDrpBus1').AsInteger;
217            BDSth2              := FieldByName('DistThreBus2').AsInteger;
218            BDSpr2              := FieldByName('DistDrpBus2').AsInteger;
219            RDSth1              := FieldByName('DistThreRes1').AsInteger;
220            RDSpr1              := FieldByName('DistDrpRes1').AsInteger;
221            UnitWtFlg           := FieldByName('FeedLineUnitW').AsInteger;
222
223            MaxLoop             := FieldByName('LoopLength').AsInteger; //RFPC0068
224            if MaxLoop < 18          //RFPC0068
225             then Maxloop := 12000   //RFPC0068
226             else MaxLoop := 18000;  //RFPC0068
227
228            Engineering_Feeder  := FieldByName('ED Feeder').AsFloat;
229            Engineering_Dist    := FieldByName('ED Distribution').AsFloat;
230            ADMNFILL            := FieldByName('ED Adm Fill').AsFloat;
231            Sharing_F_frgn_Poles:= FieldByName('SFFPoles').AsFloat;
```

```
232             Sharing_F_Aerial    := FieldByName('SFA').AsFloat;
233             Sharing_F_Buried    := FieldByName('SFB').AsFloat;
234             Sharing_F_Cond      := FieldByName('SFC').AsFloat;
235             Sharing_D_frgn_Poles:= FieldByName('SDFPoles').AsFloat;
236             Sharing_D_Aerial    := FieldByName('SDA').AsFloat;
237             Sharing_D_Buried    := FieldByName('SDB').AsFloat;
238             Sharing_D_Cond      := FieldByName('SDC').AsFloat;
239             Dropwire_Size       := FieldByName('DPWire').AsInteger;
240             OSP_Growth          := FieldByName('Growth').AsFloat;
241             OSP_Horizon         := FieldByName('Horizon').AsFloat;
242             ManHole_Spacing     := FieldByName('ManHole Spacing').AsFloat;
243             PullBox_Spacing     := FieldByName('Pull Box Spacing').AsFloat;
244             Pole_Spacing        := FieldByName('Pole Spacing').AsFloat;
245             XCONN_Factor        := FieldByName('XCONN').AsFloat;
246             Percent_GuyWire     := FieldByName('Per GuyWire').AsFloat;
247             Percent_Concrete    := FieldByName('Per Concrete').AsFloat;
248             Percent_Hand        := FieldByName('Per Hand').AsFloat;
249             PoleUsers           := FieldByName('SPU').AsFloat;
250             TrenchUsers         := FieldByName('STU').AsFloat;
251             ConduitUsers        := FieldByName('SCU').AsFloat;
252             UserDistFill        := FieldByName('Fill Dist').AsFloat;
253             UserFeedFill        := FieldByName('Fill Feeder').AsFloat;
254             UserFill            := FieldByName('UseFillFactor').AsBoolean;
255             CU_Loop_Length      := FieldByName('LoopLength').AsInteger;
256             Aer_CU_Span         := FieldByName('AerCuSpan').AsInteger;
257             Bur_CU_Span         := FieldByName('BurCuSpan').AsInteger;
258             Aer_FI_Span         := FieldByName('AerFiSpan').AsInteger;
259             Bur_FI_Span         := FieldByName('BurFiSpan').AsInteger;
260             LnFtManhole         := FieldByName('LnFtManhole').AsFloat;
261             LnFtPullBox         := FieldByName('LnFtPullBox').AsFloat;
262             WellPointDays       := FieldByName('WellPointDays').AsFloat;
263             Percent_Bore        := FieldByName('PerBoring').AsFloat;
264             MinXSize1           := FieldByName('XCSize1').AsFloat;
265             MinXSize2           := FieldByName('XCSize2').AsFloat;
266             XCMinDist           := FieldByName('MinXCDist').AsInteger;
267             Free_Drop           := FieldByName('FDrop').AsFloat;
268             Percent_Trench      := FieldByName('Per Trench').AsFloat;
269           end;
270         close; // Close outside plant runtime option table
```

```
271      end;
272
273    { Load Distribution Style table for this run }
274      TabName :=  f_associate.GetFileName(2002);
275      With DM_ICM.t_Demand do
276        begin
277          if Active then Close;
278          DatabaseName := ExtractFilePath(TabName);
279          TableName := ExtractFilename(TabName);
280          TableType := ttDefault;
281          Exclusive := False;
282          Open;
283        end;
284
285    { Create a memory table to hold Distribution Style information}
286      DS_Count := DM_ICM.t_Demand.RecordCount;
287      DS_Size  := DS_Count*SizeOf(TDistStyle);
288      GetMem (DistStyle, DS_Size);
289
290      DM_ICM.t_Demand.First;
291      Index := 1;
292    { Load Distribution Style table information }
293      while Not DM_ICM.t_Demand.EOF do
294       begin
295         DistStyle^[Index].RoadS             := DM_ICM.t_Demand.FieldByName('MinVal').AsInteger;
296         DistStyle^[Index].RoadE             := DM_ICM.t_Demand.FieldByName('MaxVal').AsInteger;
297         DistStyle^[Index].PlowFlag          := DM_ICM.t_Demand.FieldByName('PlwFlg').AsInteger;
298         DistStyle^[Index].StructFlag        := DM_ICM.t_Demand.FieldByName('StrFlg').AsInteger;
299         DistStyle^[Index].StructHighDenThld := DM_ICM.t_Demand.FieldByName('StrHDT').AsInteger;
300         DistStyle^[Index].AvgLenF           := DM_ICM.t_Demand.FieldByName('AvgLenFrac').AsFloat;
301         DistStyle^[Index].DistDmdThr        := DM_ICM.t_Demand.FieldByName('DDT').AsInteger;
302         DistStyle^[Index].DensityThreshold  := DM_ICM.t_Demand.FieldByName('ARC4DT').AsInteger;
303         for I := 1 to 8 do
304           begin
305             Str := Format('RCoef%d', [I]);
306             DistStyle^[Index].LengthCoef[I] := DM_ICM.t_Demand.FieldByName(Str).AsFloat;
307             Str := Format('DCoef%d', [I]);
308             DistStyle^[Index].DemandCoef[I] := DM_ICM.t_Demand.FieldByName(Str).AsFloat;
309           end;
```

```
310         Inc(Index);
311         DM_ICM.t_Demand.Next;
312       end;
313
314     DM_ICM.t_Demand.Close;
315
316   { Load Node table for this run }
317     TabName :=  f_associate.GetFileName(2003);
318     With DM_ICM.t_Nodes do
319       begin
320          if Active then Close;
321          DatabaseName := ExtractFilePath(TabName);
322          TableName := ExtractFilename(TabName);
323          TableType := ttDefault;
324          Exclusive := False;
325          IndexFieldNames := 'CLLI';
326          Open;
327       end;
328
329   { Load Demand table for this run }
330     TabName :=  f_associate.GetFileName(3001);
331     With DM_ICM.t_Demand do
332       begin
333          if Active then Close;
334          DatabaseName := ExtractFilePath(TabName);
335          TableName := ExtractFilename(TabName);
336          TableType := ttDefault;
337          IndexName := '';
338          IndexFieldNames := '';
339          Exclusive := False;
340          Open;
341       end;
342       Caption := 'Processing Outside Plant - Demand ['+ExtractFilename(TabName)+']';
343
344     // Create a table for feeder, backbone and distribution arcs
345     with DM_ICM.t_Arcs do
346       begin
347         Active := False;
348         DatabaseName := 'modata';
```

```
349         TableName := 'Arcfile.db';
350         TableType := ttParadox;
351         Exclusive := False;
352         with FieldDefs do
353           begin
354             Clear;
355             Add ('CLLI',         FTString,  11, True);
356             Add ('Arc Type',     FTInteger,  0, True);
357             Add ('From X',       FTInteger,  0, True);
358             Add ('From Y',       FTInteger,  0, True);
359             Add ('To X',         FTInteger,  0, True);
360             Add ('To Y',         FTInteger,  0, True);
361             Add ('Node',         FTInteger,  0, True);
362             Add ('X',            FTInteger,  0, False);
363             Add ('Y',            FTInteger,  0, False);
364             Add ('Length',       FTInteger,  0, False);
365             Add ('DistToOrigin',FTInteger,  0, False);
366             Add ('Demand',       FTInteger,  0, False);
367             Add ('LoopLenCode', FTInteger,  0, False);
368             Add ('Res Lines',    FTInteger,  0, False);
369             Add ('Bus Lines',    FTInteger,  0, False);
370             Add ('PerAerial',    FTFloat,    0, False);
371             Add ('PerBuried',    FTFloat,    0, False);
372             Add ('PerUndgnd',    FTFloat,    0, False);
373             Add ('BedRock',      FTInteger,  0, False);
374             Add ('WaterTable',   FTInteger,  0, False);
375             Add ('Elevation',    FTInteger,  0, False);
376             Add ('Highway',      FTInteger,  0, False);
377             Add ('CanPlow',      FTInteger,  0, False);
378             Add ('Structure',    FTInteger,  0, False);
379             Add ('IOF',          FTInteger,  0, False);
380             Add ('HRF',          FTInteger,  0, False);
381             Add ('IRF',          FTInteger,  0, False);   // RFPC0067
382             Add ('OSP Type',     FTString,   1, False);
383           end;
384
385         with IndexDefs do
386           begin
387             Clear;
```

```
388           Add('Main', 'CLLI;Arc Type;From X;From Y;To X;To Y;Node', [ixPrimary, ixUnique]);
389         end;
390
391       CreateTable;
392       Open;
393     end;
394
395  // Create a SAI table
396   with DM_ICM.t_DLC do
397     begin
398       Active := False;
399       DatabaseName := 'modata';
400       TableName := 'DLC.db';
401       TableType := ttParadox;
402       Exclusive := True;
403       with FieldDefs do
404         begin
405           Clear;
406           Add ('CLLI',        FTString,  11, True);
407           Add ('X',           FTInteger,  0, True);
408           Add ('Y',           FTInteger,  0, True);
409           Add ('Latitude',    FTFloat,    0, True);
410           Add ('Longitude',   FTFloat,    0, True);
411         end;
412
413       with IndexDefs do
414         begin
415           Clear;
416           Add('Main', 'CLLI;X;Y', [ixPrimary, ixUnique]);
417         end;
418
419       CreateTable;
420       Open;
421     end;
422
423  end;
424
425 procedure Tf_Demand.New1Click(Sender: TObject);
426  begin
```

```
427     sd_Demand.Filter := 'Database|*.db';
428     if  not sd_Demand.execute
429       then exit;
430     TabName := sd_Demand.FileName;
431     cb_Demand.Checked := True;
432
433     if ts_Demand.TabIndex <> 0
434       then begin
435              with DM_ICM.t_Demand do
436                begin
437                  Active := False;
438                  DatabaseName := ExtractFilePath(TabName);
439                  TableName := ExtractFilename(TabName);
440                  TableType := ttParadox;
441                  Exclusive := True;
442                  with FieldDefs do
443                    begin
444                      Clear;
445                      Add ('MinVal',          FTInteger, 0, True);
446                      Add ('MaxVal',          FTInteger, 0, True);
447                      Add ('RCoef1',          FTFloat,   0, True);
448                      Add ('RCoef2',          FTFloat,   0, True);
449                      Add ('RCoef3',          FTFloat,   0, True);
450                      Add ('RCoef4',          FTFloat,   0, True);
451                      Add ('RCoef5',          FTFloat,   0, True);
452                      Add ('RCoef6',          FTFloat,   0, True);
453                      Add ('RCoef7',          FTFloat,   0, True);
454                      Add ('RCoef8',          FTFloat,   0, True);
455                      Add ('DCoef1',          FTFloat,   0, True);
456                      Add ('DCoef2',          FTFloat,   0, True);
457                      Add ('DCoef3',          FTFloat,   0, True);
458                      Add ('DCoef4',          FTFloat,   0, True);
459                      Add ('DCoef5',          FTFloat,   0, True);
460                      Add ('DCoef6',          FTFloat,   0, True);
461                      Add ('DCoef7',          FTFloat,   0, True);
462                      Add ('DCoef8',          FTFloat,   0, True);
463                      Add ('PlwFlg',          FTinteger, 0, True);
464                      Add ('StrFlg',          FTinteger, 0, True);
465                      Add ('StrHDT',          FTinteger, 0, True);
```

```
466                     Add ('AvgLenFrac',       FTFloat,   0, False);
467                     Add ('DDT',              FTInteger, 0, False);
468                     Add ('ARC4DT',           FTInteger, 0, False);
469                   end;
470
471                 with IndexDefs do
472                   begin
473                     Clear;
474                     Add('Std', 'MinVal', [ixPrimary, ixUnique]);
475                   end;
476
477                 CreateTable;
478                 Open;
479               end;
480             end;
481   Caption := 'Demand ['+ExtractFilename(TabName)+']';
482 end;
483
484 procedure Tf_Demand.Exit1Click(Sender: TObject);
485 begin
486   if DM_ICM.t_Demand.Active
487     then DM_ICM.t_Demand.Close;
488
489   if cb_Demand.Checked
490     then case Mode of
491           1 : f_associate.SetFileName(3001, TabName);
492           2 : f_associate.SetFileName(2002, TabName);
493         end;
494
495   if RunFlag then f_associate.SetFileName(100, 'YES');
496   Hide;
497 end;
498
499 procedure Tf_Demand.Open1Click(Sender: TObject);
500 begin
501   od_Demand.Filter := 'Database|*.db';
502   if  not od_Demand.execute
503     then exit;
504
```

```
505     cb_Demand.Checked := False;
506
507    TabName := od_Demand.FileName;
508
509    with DM_ICM.t_Demand do
510       begin
511         Active := False;
512         DatabaseName := ExtractFilePath(TabName);
513         TableName := ExtractFilename(TabName);
514         TableType := ttDefault;
515         Exclusive := False;
516         Open;
517       end;
518
519     Caption := 'Demand ['+ExtractFilename(TabName)+']';
520   end;
521
522   procedure Tf_Demand.FormShow(Sender: TObject);
523   begin
524     cb_Demand.Checked := False;
525     l_Copyright.Caption := f_associate.GetFileName(5);
526     p_Progress.Visible   := False;
527
528     Case Mode of
529       1 : begin
530             Width   := 440;
531             Height  := 320;
532             Caption := 'Demand Data';
533           end;
534       2 : begin
535             Width                 := 440;
536             Height                := 320;
537             ts_Demand.TabIndex    := 1;
538             db_Demand.DataSource  := ds_Demand;
539             ts_Demand.Visible     := True;
540             db_Demand.Visible     := True;
541             cb_Demand.Visible     := True;
542             New1.Enabled          := True;
543             Import1.Enabled       := True;
```

```
544             Export1.Enabled       := True;
545             TabName :=  f_associate.GetFileName(2002);
546             if TabName = ''
547               then begin
548                      od_Demand.Filter := 'Database|*.db';
549                      if not od_Demand.Execute
550                        then exit;
551                      TabName := od_Demand.FileName;
552                    end;
553              With DM_ICM.t_Demand do
554                begin
555                  if Active then Close;
556                  DatabaseName := ExtractFilePath(TabName);
557                  TableName := ExtractFilename(TabName);
558                  TableType := ttParadox;
559                  Exclusive := False;
560                  Open;
561                end;
562             Caption := 'Distribution Style ['+ExtractFilename(TabName)+']';
563           end;
564       3 : begin
565             Width := 400;
566             Height:= 150;
567             p_Progress.Visible   := True;
568             ts_Demand.Visible    := False;
569             db_Demand.Visible    := False;
570             cb_Demand.Visible    := False;
571             DBNavigator1.Visible := False;
572           end;
573     end;
574
575     FormResize(Sender);
576   end;
577
578   procedure Tf_Demand.FormResize(Sender: TObject);
579   begin
580     cb_Demand.Top := ClientHeight - (cb_Demand.Height+5);
581     cb_Demand.Left := (ClientWidth-cb_Demand.width) div 2;
582     DBNavigator1.Top := cb_Demand.Top-(DBNavigator1.Height+5);
```

```
583     DBNavigator1.Left := (ClientWidth-DBNavigator1.width) div 2;
584     p_Progress.Top := l_Copyright.Height + 5;
585     db_Demand.Top  := l_Copyright.Height + 5;
586     db_Demand.Left := 5;
587     db_Demand.Height  := DBNavigator1.Top-db_Demand.Top-(ts_Demand.Height+5);
588     db_Demand.Width  := ClientWidth-10;
589
590     ts_Demand.Width  := db_Demand.Width;
591     ts_Demand.Top     := db_Demand.Top + db_Demand.Height;
592
593     p_Progress.Left  := (ClientWidth - p_Progress.Width) div 2;
594     ts_Demand.Left    := 5;
595
596     DBNavigator1.Visible:= True;
597     case mode of
598      1 : db_Demand.Visible  := True;
599      2 : db_Demand.Visible  := False;
600      3 : DBNavigator1.Visible:= False;
601     end;
602
603     case ts_Demand.TabIndex of
604      1 : begin
605            db_Demand.Visible  := True;
606            ts_Demand.Top      := db_Demand.Top + db_Demand.Height;
607            ts_Demand.Width    := db_Demand.Width;
608          end;
609     end;
610   end;
611
612   procedure Tf_Demand.ts_DemandChange(Sender: TObject; NewTab: Integer;
613     var AllowChange: Boolean);
614   begin
615     db_Demand.Visible  := False;
616
617     if (NewTab > 0) and (ts_Demand.TabIndex = 0) then begin
618       Mode := 2;
619       if DM_ICM.t_Demand.Active then DM_ICM.t_Demand.close;
620       if cb_Demand.Checked then begin
621         f_associate.SetFileName(3001, TabName);  {grid.db: grid table}
```

```
622         cb_Demand.Checked := False;
623       end;
624
625       TabName :=  f_associate.GetFileName(2002);   {gs.db: gridstyle table}
626       if TabName = '' then begin
627         od_Demand.Filter := 'Database|*.db';
628         if not od_Demand.Execute then
629           exit;
630         TabName := od_Demand.FileName;
631       end;
632       With DM_ICM.t_Demand do begin
633         DatabaseName := ExtractFilePath(TabName);
634         TableName := ExtractFilename(TabName);
635         TableType := ttParadox;
636         Exclusive := False;
637         IndexName := '';
638         IndexFieldNames := '';
639         Open;
640       end;
641       Caption := 'Distribution  Style ['+ExtractFilename(TabName)+']';
642     end;
643
644     if (NewTab = 0) and (ts_Demand.TabIndex > 0) then begin
645       mode := 1;
646       if DM_ICM.t_Demand.Active then DM_ICM.t_Demand.close;
647       if cb_Demand.Checked then begin
648         f_associate.SetFileName(2002, TabName);
649         cb_Demand.Checked := False;
650       end;
651
652       TabName :=  f_associate.GetFileName(3001);   {grid.db: grid table}
653       if TabName = '' then begin
654         od_Demand.Filter := 'Database|*.db';
655         if not od_Demand.Execute then exit;
656         TabName := od_Demand.FileName;
657       end;
658       With DM_ICM.t_Demand do begin
659         DatabaseName := ExtractFilePath(TabName);
660         TableName := ExtractFilename(TabName);
```

```
661         TableType := ttParadox;
662         IndexName := '';
663         IndexFieldNames := '';
664         Exclusive := False;
665         Open;
666       end;
667       Caption := 'Demand Data ['+ExtractFilename(TabName)+']';
668     end;
669
670   Tab1.Visible := True;
671   case NewTab of
672     0: begin
673           db_Demand.Visible  := True;
674           ts_Demand.Top      := db_Demand.Top + db_Demand.Height;
675           ts_Demand.Width    := db_Demand.Width;
676           Mode               := 1;
677           New1.Enabled       := False;
678           Import1.Enabled    := False;
679           Export1.Enabled    := False;
680           Tab1.Caption       := ts_Demand.Tabs[Newtab];
681        end;
682     1: begin
683          db_Demand.Visible   := True;
684          New1.Enabled        := True;
685          Import1.Enabled     := True;
686          Export1.Enabled     := True;
687          Tab1.Caption        := ts_Demand.Tabs[Newtab];
688        end;
689   end;
690 end;
691
692 procedure Tf_Demand.FormClose(Sender: TObject; var Action: TCloseAction);
693 begin
694   if DM_ICM.t_Demand.Active then
695     DM_ICM.t_Demand.Close;
696 end;
697
698 procedure Tf_Demand.Print1Click(Sender: TObject);
699 begin
```

```
700     GenericPrintTable(DM_ICM.t_Demand, Nil, 0, False, 'Demand Table');
701   end;
702
703   procedure Tf_Demand.Export1Click(Sender: TObject);
704   begin
705     ExportTable(DM_ICM.t_Demand);
706   end;
707
708   procedure Tf_Demand.PrintDemand( Open : Boolean);
709    begin
710      if Open
711        then begin
712               TabName :=  f_associate.GetFileName(3001);
713               With DM_ICM.t_Demand do
714                begin
715                  DatabaseName := ExtractFilePath(TabName);
716                  TableName := ExtractFilename(TabName);
717                  TableType := ttParadox;
718                  Exclusive := False;
719                  Open;
720                end;
721             end;
722      GenericPrintTable(DM_ICM.t_Demand, Nil, 0, False, 'Demand Table');
723      if Open then DM_ICM.t_Demand.Close;
724    end;
725
726   procedure Tf_Demand.FormActivate(Sender: TObject);
727   begin
728     UnitName := 'Demand';
729   end;
730
731   procedure Tf_Demand.Tab1Click(Sender: TObject); // Help file connection
732   begin
733     Case ts_Demand.TabIndex of
734       0 : GoHelp ('Datatables.hlp', 433);
735       1 : GoHelp ('Datatables.hlp', 444);
736       2 : GoHelp ('Howdoi.hlp', 221);
737     end;
738   end;
```

```
procedure Tf_Demand.Common1Click(Sender: TObject); // Help file connection
begin
 GoHelp ('MenuIt.hlp', 323);
end;

procedure Tf_Demand.Import1Click(Sender: TObject);
begin
 DM_ICM.t_Demand.Close;
 ImportDataCall('', AliasPath+'\'+DM_ICM.t_Demand.Tablename, 10);
 DM_ICM.t_Demand.Open;
end;

end.
```

DISTRIB.pas

```
2
3   unit Distrib;
4
5   { RFPC 0044  - Improve the 4-wire NID Cost.
6     Originated 6/11/99
7
8     The change will improve how the NID is being costed in association with 4-wire
9     loops.  Currently, the material item being utilized for costing the smallest
10    NID (6 drops and below) is a casing equipped with one protector.  The casing
11    can handle up to 6 protectors.  As part of mapping a 4-wire unbundled loop,
12    the NID cost is multiplied by two.  In the case of the 6 pair NID, the case is
13    being included twice when it should only be included once (when using the
14    default drop size of 3 pair).
15  }
16
17
18  interface
19   procedure Distribution;
20
21  implementation
22   uses
23     Math, ICMDM, OSPTypes, SysUtils, dbtables, Dialogs, Invntry,
24     Osplabor, Ospmater;
25
26
27  procedure Distribution;
28   Var
29     Terminal      : Integer;
30     RNID          : Integer;
31     BNID          : Integer;
32     DropLen       : Integer;
33     RDropSize     : Integer;
34     BDropSize     : Integer;
35     Res_Units     : Integer;
36     Bus_Units     : Integer;
```

```
37      Loops         : Integer;
38      AvgLength     : Integer;
39      Index         : Integer;
40      N, I          : Integer;
41
42      BusF          : Double;     // Bus portion of the total demand
43      Drop_Util     : Double;     // Drop utilization
44      TInv          : Double;     // Total investment
45      RInv          : Double;     // Res investment
46      BInv          : Double;     // Bus investment
47      DSM           : Double;     // Demand point's square miles / Density
48      RFA           : Double;     // Road footage adjustment
49      T4Inv         : Double;     // Total investment 4-wire  RFPC 0044
50      R4Inv         : Double;     // Res investment 4-wire  RFPC 0044
51      B4Inv         : Double;     // Bus investment 4-wire  RFPC 0044
52
53    begin
54      // if no demand don't process the distribution
55      if (DistValues.Res_Lines +DistValues.Bus_Lines) < 1
56        then exit;
57
58      // Classify Distribution style }
59      DS_index := 0;
60      for N := 1 to DS_Count do
61        begin
62          if (DistValues.road_feet>=DistStyle^[N].RoadS) and (DistValues.road_feet<=DistStyle^[N].RoadE)
63            then DS_Index := N;
64        end;
65
66      if (DS_Index = 0)
67        then  Exit;
68
69      Loops     := Round(DistValues.Res_Lines + DistValues.Bus_Lines);
70      Drop_Util := 0.50;
71
72      if Loops < DistStyle^[DS_Index].DistDmdThr then RFA:=0.0
73      else
74        begin
75          // Calculate the square miles of a demand point and
```

```
76          // adjust loops per square mile to loops per demand point.
77          DSM := Round(((DM_ICM.t_Nodes.FieldByName('Width Demand Pt').AsFloat *
78                  DM_ICM.t_Nodes.FieldByName('Height Demand Pt').AsFloat) / sqr(5280.0))*
79                  DistStyle^[DS_Index].DensityThreshold);
80          if Loops >= DSM
81            then RFA := 1.0
82            else RFA := RoadFtAdj;
83        end;
84      AvgLength := round(DistStyle^[DS_Index].AvgLenF * DistValues.road_feet * RFA * Loops);
85
86      // Calculate number of units
87      Res_Units := Ceil( DistValues.Res_Lines / Asmp.RLnPUnit);
88      Bus_Units := Ceil( DistValues.Bus_Lines / Asmp.BLnPUnit);
89
90      // Nid and Drop calculation
91      RNID := Asmp.Dropwire_Size;
92      BNID := Asmp.Dropwire_Size;
93      if Res_Units > Asmp.RDSth1  then RNID := Asmp.RDSpr1 ;
94      if Bus_Units > Asmp.BDSth1  then BNID := Asmp.BDSpr1;
95      if Bus_Units > Asmp.BDSth2  then BNID := Asmp.BDSpr2;
96      RDropSize := RNID;
97      BDropSize := BNID;
98
99      // Drop length calculation
100     DropLen := Round(Sqrt(1350000.0/(Res_Units + Bus_Units)));
101     if DropLen < Asmp.MnDrpLn then DropLen := Asmp.MnDrpLn;
102     if DropLen > Asmp.MxDrpLn then DropLen := Asmp.MxDrpLn;
103
104     // Terminals calculation
105     Terminal := 0;
106     if Res_Units < 500
107       then Terminal := Terminal + Ceil(Res_Units / Asmp.RUnitPTerm);
108     if Bus_Units < 500
109       then Terminal := Terminal + Ceil(Bus_Units / Asmp.BUnitPTerm);
110
111     // Bus portion of the total demand
112     BusF :=  DistValues.Bus_Lines / Loops;
113
114   // NID - based on size of Network Interface Device (NID) which
```

```
115     // equals the size of the drop
116       case RNID of        // residential NID
117        1..6  : begin
118                  // RFPC 0044
119                  if (Asmp.RLnPUnit <= 6) then
120                    begin
121                      RInv := (f_OSPMaterial.Mat('NID',1)+f_OSPLabor.Labor('LS14B',  Tier_Flag ))* Res_Units +
122                               f_OSPMaterial.Mat('NIDPROT',1)*DistValues.Res_Lines;
123                      PartInventory.Write (DistValues.Clli, 'D', 'NID', 1, Res_Units,
124                                           DistValues.Cluster.X, DistValues.Cluster.Y);
125                    end
126                  else
127                    begin
128                      RInv := (f_OSPMaterial.Mat('NID',12) + (f_OSPMaterial.Mat('NIDPROT',12)*12) +
129                          f_OSPLabor.Labor('LS13A',  Tier_Flag ))* Res_Units;
130                      PartInventory.Write (DistValues.Clli, 'D', 'NID', 12, Res_Units,
131                                           DistValues.Cluster.X, DistValues.Cluster.Y);
132                    end;
133                  // RFPC 0044
134                  // 6 pair NID 4-wire investment where Asmp.RLnPUnit <= 3
135                  if (Asmp.RLnPUnit <= 3) then
136                    begin
137                      R4Inv := (f_OSPMaterial.Mat('NID',1)+f_OSPLabor.Labor('LS14B',  Tier_Flag ))* Res_Units
138     +
139                               f_OSPMaterial.Mat('NIDPROT',1)*DistValues.Res_Lines*2;
140                      PartInventory.Write (DistValues.Clli, 'D', 'NID4', 1, Res_Units,
141                                           DistValues.Cluster.X, DistValues.Cluster.Y);
142                    end
143                  else
144                    begin
145                      R4Inv := (f_OSPMaterial.Mat('NID',12) + (f_OSPMaterial.Mat('NIDPROT',12)*12) +
146                          f_OSPLabor.Labor('LS13A',  Tier_Flag ))* Res_Units;
147                      PartInventory.Write (DistValues.Clli, 'D', 'NID4', 12, Res_Units,
148                                           DistValues.Cluster.X, DistValues.Cluster.Y);
149                    end;
150                end;
151        7..12 : begin
152                  // RFPC 0044
153                  RInv := (f_OSPMaterial.Mat('NID',12) + (f_OSPMaterial.Mat('NIDPROT',12)*12) +
```

```
154                         f_OSPLabor.Labor('LS13A',  Tier_Flag )) * Ceil(DistValues.Res_Lines/(Drop_Util *
155   RDropSize));
156                   PartInventory.Write (DistValues.Clli, 'D', 'NID', 12, Ceil(DistValues.Res_Lines/(Drop_Util *
157   RDropSize)),
158                                       DistValues.Cluster.X, DistValues.Cluster.Y);
159                   // RFPC 0044
160                   // 12 pair NID 4-wire investment
161                   R4Inv := (f_OSPMaterial.Mat('NID',25) + (f_OSPMaterial.Mat('NIDPROT',25)*25) +
162                             f_OSPLabor.Labor('LS13A',  Tier_Flag )) *
163                            Ceil(DistValues.Res_Lines/(Drop_Util * RDropSize));
164                   PartInventory.Write (DistValues.Clli, 'D', 'NID4', 25, Ceil(DistValues.Res_Lines/(Drop_Util
165   * RDropSize)),
166                                       DistValues.Cluster.X, DistValues.Cluster.Y);
167                 end;
168         13..25: begin
169                   // RFPC 0044
170                   RInv := (f_OSPMaterial.Mat('NID',25) + (f_OSPMaterial.Mat('NIDPROT',25)*25) +
171                            f_OSPLabor.Labor('LS13A',  Tier_Flag )) *
172                           Ceil(DistValues.Res_Lines/(Drop_Util * RDropSize));
173                   PartInventory.Write (DistValues.Clli, 'D', 'NID', 25, Ceil(DistValues.Res_Lines/(Drop_Util *
174   RDropSize)),
175                                       DistValues.Cluster.X, DistValues.Cluster.Y);
176                   // RFPC 0044
177                   // 25 pair NID 4-wire investment
178                   R4Inv := (f_OSPMaterial.Mat('NID',50) + (f_OSPMaterial.Mat('NIDPROT',50)*50) +
179                             f_OSPLabor.Labor('LS13A',  Tier_Flag )) *
180                            Ceil(DistValues.Res_Lines/(Drop_Util * RDropSize));
181                   PartInventory.Write (DistValues.Clli, 'D', 'NID4', 50, Ceil(DistValues.Res_Lines/(Drop_Util
182   * RDropSize)),
183                                       DistValues.Cluster.X, DistValues.Cluster.Y);
184                 end;
185           else  begin
186                   RInv := (f_OSPMaterial.Mat('NID',50) + (f_OSPMaterial.Mat('NIDPROT',50)*50) +
187                            f_OSPLabor.Labor('LS13A',  Tier_Flag )) *
188                           Ceil(DistValues.Res_Lines/(Drop_Util * RDropSize));
189                   PartInventory.Write (DistValues.Clli, 'D', 'NID', 50, Ceil(DistValues.Res_Lines/(Drop_Util *
190   RDropSize)),
191                                       DistValues.Cluster.X, DistValues.Cluster.Y);
192                   R4Inv := RInv * 2;
```

```
193                 PartInventory.Write (DistValues.Clli, 'D', 'NID4', 50, Ceil(DistValues.Res_Lines/(Drop_Util
194  • RDropSize))*2,
195                                      DistValues.Cluster.X, DistValues.Cluster.Y);
196               end;
197      end;
198
199      case BNID of  // business NID
200       1..6  : begin
201                 // RFPC 0044
202                 if (Asmp.BLnPUnit <= 6) then
203                   begin
204                     BInv := (f_OSPMaterial.Mat('NID',1)+f_OSPLabor.Labor('LS14B',  Tier_Flag ))* Bus_Units +
205                             f_OSPMaterial.Mat('NIDPROT',1)*DistValues.Bus_Lines;
206                     PartInventory.Write (DistValues.Clli, 'D', 'NID', 1, Bus_Units,
207                                          DistValues.Cluster.X, DistValues.Cluster.Y);
208                   end
209                 else
210                   begin
211                     BInv := (f_OSPMaterial.Mat('NID',12) + (f_OSPMaterial.Mat('NIDPROT',12)*12) +
212                         f_OSPLabor.Labor('LS13A',  Tier_Flag ))* Bus_Units;
213                     PartInventory.Write (DistValues.Clli, 'D', 'NID', 12, Bus_Units,
214                                          DistValues.Cluster.X, DistValues.Cluster.Y);
215                   end;
216                 // RFPC 0044
217                 // 6 pair NID 4-wire investment where Asmp.BLnPUnit <= 3
218                 if (Asmp.BLnPUnit <= 3) then
219                   begin
220                     B4Inv := (f_OSPMaterial.Mat('NID',1)+f_OSPLabor.Labor('LS14B',  Tier_Flag ))* Bus_Units
221  +
222                              f_OSPMaterial.Mat('NIDPROT',1)*DistValues.Bus_Lines*2;
223                     PartInventory.Write (DistValues.Clli, 'D', 'NID4', 1, Bus_Units,
224                                          DistValues.Cluster.X, DistValues.Cluster.Y);
225                   end
226                 else
227                   begin
228                     B4Inv := (f_OSPMaterial.Mat('NID',12) + (f_OSPMaterial.Mat('NIDPROT',12)*12) +
229                         f_OSPLabor.Labor('LS13A',  Tier_Flag ))* Bus_Units;
230                     PartInventory.Write (DistValues.Clli, 'D', 'NID4', 12, Bus_Units,
231                                          DistValues.Cluster.X, DistValues.Cluster.Y);
```

```
232                 end;
233             end;
234       7..12 : begin
235               // RFPC 0044
236               BInv := (f_OSPMaterial.Mat('NID',12) + (f_OSPMaterial.Mat('NIDPROT',12)*12) +
237                        f_OSPLabor.Labor('LS13A',  Tier_Flag )) *
238                       Ceil(DistValues.Bus_Lines/(Drop_Util * BDropSize));
239               PartInventory.Write (DistValues.Clli, 'D', 'NID', 12, Ceil(DistValues.Bus_Lines/
240                                    (Drop_Util * BDropSize)), DistValues.Cluster.X, DistValues.Cluster.Y);
241               // RFPC 0044
242               // 12 pair NID 4-wire investment
243               B4Inv := (f_OSPMaterial.Mat('NID',25) + (f_OSPMaterial.Mat('NIDPROT',25)*25) +
244                         f_OSPLabor.Labor('LS13A',  Tier_Flag )) *
245                        Ceil(DistValues.Bus_Lines/(Drop_Util * BDropSize));
246               PartInventory.Write (DistValues.Clli, 'D', 'NID4', 25, Ceil(DistValues.Bus_Lines/
247                                    (Drop_Util * BDropSize)), DistValues.Cluster.X, DistValues.Cluster.Y);
248             end;
249       13..25: begin
250               // RFPC 0044
251               BInv := (f_OSPMaterial.Mat('NID',25) + (f_OSPMaterial.Mat('NIDPROT',25)*25) +
252                        f_OSPLabor.Labor('LS13A',  Tier_Flag )) *
253                       Ceil(DistValues.Bus_Lines/(Drop_Util * BDropSize));
254               PartInventory.Write (DistValues.Clli, 'D', 'NID', 25, Ceil(DistValues.Bus_Lines/
255                                    (Drop_Util * BDropSize)), DistValues.Cluster.X, DistValues.Cluster.Y);
256               // RFPC 0044
257               // 25 pair NID 4-wire investment
258               B4Inv := (f_OSPMaterial.Mat('NID',50) + (f_OSPMaterial.Mat('NIDPROT',50)*50) +
259                         f_OSPLabor.Labor('LS13A',  Tier_Flag )) *
260                        Ceil(DistValues.Bus_Lines/(Drop_Util * BDropSize));
261               PartInventory.Write (DistValues.Clli, 'D', 'NID4', 50, Ceil(DistValues.Bus_Lines/
262                                    (Drop_Util * BDropSize)), DistValues.Cluster.X, DistValues.Cluster.Y);
263             end;
264       else  begin
265               // RFPC 0044
266               BInv := (f_OSPMaterial.Mat('NID',50) + (f_OSPMaterial.Mat('NIDPROT',50)*50) +
267                        f_OSPLabor.Labor('LS13A',  Tier_Flag )) *
268                       Ceil(DistValues.Bus_Lines/(Drop_Util * BDropSize));
269               PartInventory.Write (DistValues.Clli, 'D', 'NID', 50, Ceil(DistValues.Bus_Lines/
270                                    (Drop_Util * BDropSize)), DistValues.Cluster.X, DistValues.Cluster.Y);
```

```
271                B4Inv := BInv * 2;
272                PartInventory.Write (DistValues.Clli, 'D', 'NID4', 50, Ceil(DistValues.Bus_Lines/
273                                     (Drop_Util * BDropSize))*2, DistValues.Cluster.X, DistValues.Cluster.Y);
274             end;
275      end;
276
277     { Ouput NID investments. }
278     TInv := BInv + RInv;
279     // RFPC 0044
280     T4Inv := B4Inv + R4Inv;
281     if (DistValues.Cluster.X=0) and (DistValues.Cluster.Y=0)
282      then begin
283             OSP_Table.Write (DistValues.CLLI, '242310', 'TDCNID', 'D', TInv,  TInv,  0.0);
284             OSP_Table.Write (DistValues.CLLI, '242310', 'RDCNID', 'D', RInv,  RInv,  0.0);
285             OSP_Table.Write (DistValues.CLLI, '242310', 'BDCNID', 'D', BInv,  BInv,  0.0);
286             //RFPC 0044
287             OSP_Table.Write (DistValues.CLLI, '242310', 'TDCNID4', 'D', T4Inv,  T4Inv,  0.0);
288             OSP_Table.Write (DistValues.CLLI, '242310', 'RDCNID4', 'D', R4Inv,  R4Inv,  0.0);
289             OSP_Table.Write (DistValues.CLLI, '242310', 'BDCNID4', 'D', B4Inv,  B4Inv,  0.0);
290          end
291     else begin
292             OSP_Table.Write (DistValues.CLLI, '242310', 'TDFNID', 'D', TInv,  TInv,   0.0);
293             OSP_Table.Write (DistValues.CLLI, '242310', 'RDFNID', 'D', RInv,  RInv,   0.0);
294             OSP_Table.Write (DistValues.CLLI, '242310', 'BDFNID', 'D', BInv,  BInv,   0.0);
295             // RFPC 0044
296             OSP_Table.Write (DistValues.CLLI, '242310', 'TDFNID4', 'D', T4Inv,  T4Inv,   0.0);
297             OSP_Table.Write (DistValues.CLLI, '242310', 'RDFNID4', 'D', R4Inv,  R4Inv,   0.0);
298             OSP_Table.Write (DistValues.CLLI, '242310', 'BDFNID4', 'D', B4Inv,  B4Inv,   0.0);
299          end;
300
301     {Terminals }
302     TInv := (f_OSPMaterial.Mat('TERM', 1) + f_OSPLabor.Labor('LS13A',  Tier_Flag ) +
303              f_OSPLabor.Labor('LP61A',  Tier_Flag )) * Terminal;
304     BInv := TInv * Busf;
305     RInv := Tinv - BInv;
306
307     PartInventory.Write (DistValues.Clli, 'D', 'TERM', 1, Terminal,
308                          DistValues.Cluster.X, DistValues.Cluster.Y);
309
```

```
310      { Ouput Terminal investments. }
311      if (DistValues.Cluster.X=0) and (DistValues.Cluster.Y=0)
312       then begin
313             OSP_Table.Write (DistValues.CLLI, '242310', 'TDCTERM', 'D', TInv,  TInv,  0.0);
314             OSP_Table.Write (DistValues.CLLI, '242310', 'RDCTERM', 'D', RInv,  RInv,  0.0);
315             OSP_Table.Write (DistValues.CLLI, '242310', 'BDCTERM', 'D', BInv,  BInv,  0.0);
316           end
317       else begin
318             OSP_Table.Write (DistValues.CLLI, '242310', 'TDFTERM', 'D', TInv,  TInv,  0.0);
319             OSP_Table.Write (DistValues.CLLI, '242310', 'RDFTERM', 'D', RInv,  RInv,  0.0);
320             OSP_Table.Write (DistValues.CLLI, '242310', 'BDFTERM', 'D', BInv,  BInv,  0.0);
321           end;
322
323      {DROP  CALCULATION - for residential and business drops }
324      if RDropSize > 5    // residential drop >5 pairs (Res threshold input) (Res drop pair input)
325        then begin
326              RInv := (f_OSPMaterial.Mat('DWIRE', RDropSize) +
327                       f_OSPLabor.Labor('LP51A',  Tier_Flag ))* DropLen;
328              RInv := RInv * Ceil(DistValues.Res_Lines /(Drop_Util * RDropSize));
329              PartInventory.Write (DistValues.Clli, 'D', 'DWIRE', RDropSize, Ceil(DistValues.Res_Lines /
330                                   (Drop_Util * RDropSize ))*DropLen, DistValues.Cluster.X,
331   DistValues.Cluster.Y);
332            end
333            // residential drop <=5 pairs
334        else begin        // residential drop <=5 pairs
335              RInv := (f_OSPMaterial.Mat('DWIRE', RDropSize) +
336                       f_OSPLabor.Labor('LP50A',  Tier_Flag ) * (1.0-Asmp.Free_Drop))* DropLen;
337              RInv := RInv * Res_Units;
338              PartInventory.Write (DistValues.Clli, 'D', 'DWIRE', RDropSize, Round(Res_Units * DropLen),
339                                   DistValues.Cluster.X, DistValues.Cluster.Y);
340              {Free drop placement is based on the subdivision developer providing
341              the drop trench at no cost. Used only in residential developments.}
342            end;
343
344      if BDropSize > 5    // business drop >5 pairs (Bus threshold input) (Bus drop pair input)
345        then begin
346              BInv := (f_OSPMaterial.Mat('DWIRE', BDropSize) +
347                       f_OSPLabor.Labor('LP51A',  Tier_Flag ))* DropLen;
348              BInv := BInv * Ceil(DistValues.Bus_Lines /(Drop_Util * BDropSize ));
```

```
349               PartInventory.Write (DistValues.Clli, 'D', 'DWIRE', BDropSize, Ceil(DistValues.Bus_Lines /
350                                    (Drop_Util * BDropSize ))*DropLen, DistValues.Cluster.X,
351   DistValues.Cluster.Y)
352             end
353             // business drop <=5 pairs
354         else begin
355               BInv := (f_OSPMaterial.Mat('DWIRE', BDropSize) +
356                        f_OSPLabor.Labor('LP50A',  Tier_Flag ))* DropLen;
357               BInv := BInv * Bus_Units;
358               PartInventory.Write (DistValues.Clli, 'D', 'DWIRE', BDropSize, Round(DropLen * Bus_Units),
359                                    DistValues.Cluster.X, DistValues.Cluster.Y);
360             end;
361
362     { Ouput Drop investments. }
363     TInv := BInv + RInv;
364     if (DistValues.Cluster.X=0) and (DistValues.Cluster.Y=0)
365      then begin
366             OSP_Table.Write (DistValues.CLLI, '242310', 'TDCDROPCU', 'D', TInv,  TInv,  0.0);
367             OSP_Table.Write (DistValues.CLLI, '242310', 'RDCDROPCU', 'D', RInv,  RInv,  0.0);
368             OSP_Table.Write (DistValues.CLLI, '242310', 'BDCDROPCU', 'D', BInv,  BInv,  0.0);
369           end
370      else begin
371             OSP_Table.Write (DistValues.CLLI, '242310', 'TDFDROPCU', 'D', TInv,  TInv,  0.0);
372             OSP_Table.Write (DistValues.CLLI, '242310', 'RDFDROPCU', 'D', RInv,  RInv,  0.0);
373             OSP_Table.Write (DistValues.CLLI, '242310', 'BDFDROPCU', 'D', BInv,  BInv,  0.0);
374           end;
375
376     if Loops < DistStyle^[DS_Index].DistDmdThr
377       then exit;
378
379     // Calculate the square miles of a demand point.
380     DSM := (DM_ICM.t_Nodes.FieldByName('Width Demand Pt').AsFloat *
381             DM_ICM.t_Nodes.FieldByName('Height Demand Pt').AsFloat) / sqr(5280.0);
382
383     // Adjust loops per square mile to loops per demand point.
384     DSM := round(DSM * DistStyle^[DS_Index].DensityThreshold);
385     if Loops >= DSM
386       then RFA := 1.0
387       else RFA := RoadFtAdj;
```

```
388
389     // process Demand result
390      for Index := 1 to 8 do
391        begin
392          if (Round(Loops * DistStyle^[DS_Index].DemandCoef[Index]) > 0) and
393             (Round(DistValues.road_feet * RFA *
394                    DistStyle^[DS_Index].LengthCoef[Index]) > 0)
395            then begin
396                   DM_ICM.t_Arcs.Append;
397                   DM_ICM.t_Arcs.FieldByName('CLLI').AsString       := DistValues.Clli;
398                   DM_ICM.t_Arcs.FieldByName('Arc Type').AsInteger  := 4;
399                   DM_ICM.t_Arcs.FieldByName('X').AsInteger         := DistValues.Cluster.X;
400                   DM_ICM.t_Arcs.FieldByName('Y').AsInteger         := DistValues.Cluster.Y;
401                   DM_ICM.t_Arcs.FieldByName('From X').AsInteger    := DistValues.Demand.X;
402                   DM_ICM.t_Arcs.FieldByName('From Y').AsInteger    := DistValues.Demand.Y;
403                   DM_ICM.t_Arcs.FieldByName('To X').AsInteger      := DistValues.Demand.X;
404                   DM_ICM.t_Arcs.FieldByName('To Y').AsInteger      := DistValues.Demand.Y;
405                   DM_ICM.t_Arcs.FieldByName('Node').AsInteger      := Index;
406                   DM_ICM.t_Arcs.FieldByName('Length').AsFloat      := DistValues.road_feet * RFA *
407                                                                       DistStyle^[DS_Index].LengthCoef[Index];
408                   DM_ICM.t_Arcs.FieldByName('DistToOrigin').AsInteger := Index;
409                   I   := Round(Loops * DistStyle^[DS_Index].DemandCoef[Index]);
410                   DM_ICM.t_Arcs.FieldByName('Demand').AsFloat       := I;
411                   DM_ICM.t_Arcs.FieldByName('LoopLenCode').AsInteger:= round( AvgLength *
412
413   DistStyle^[DS_Index].LengthCoef[Index]);
414                   DM_ICM.t_Arcs.FieldByName('Bus Lines').AsFloat    := I * BusF;
415                   DM_ICM.t_Arcs.FieldByName('Res Lines').AsFloat    := I-DM_ICM.t_Arcs.FieldByName('Bus
416   Lines').AsInteger;
417                   DM_ICM.t_Arcs.FieldByName('BedRock').AsInteger    := BedRock;
418                   DM_ICM.t_Arcs.FieldByName('WaterTable').AsInteger := WaterTable;
419                   DM_ICM.t_Arcs.FieldByName('Elevation').AsInteger  := Elevation;
420                   DM_ICM.t_Arcs.FieldByName('Highway').AsInteger    := Highway;
421                   DM_ICM.t_Arcs.FieldByName('CanPlow').AsInteger    := DistStyle^[DS_Index].PlowFlag;
422                   if Bus_units > DistStyle^[DS_Index].StructHighDenThld
423                     then DM_ICM.t_Arcs.FieldByName('Structure').AsInteger  := 2
424                     else DM_ICM.t_Arcs.FieldByName('Structure').AsInteger  :=
425   DistStyle^[DS_Index].StructFlag;
426                   DM_ICM.t_Arcs.FieldByName('IOF').AsInteger        := 0;
```

```
427                   DM_ICM.t_Arcs.FieldByName('HRF').AsInteger        := 0;
428                   DM_ICM.t_Arcs.FieldByName('IRF').AsInteger        := 0;  // RFPC0067
429                   DM_ICM.t_Arcs.Post;
430                 end;
431         end;
432     end;
433
434   end.
435
```

EXPENSE.pas

```
unit Expense;
{
  This unit contains the routines to maintain the data required for expenses.
}

{ RFPC 0051  - Rename all occurrences of the field "Other Direct", included in
               the Expense Module ICM source code and the Expense Module base
               table column description, to "Direct Other" and rename the column
               headings found in the 'Expense Results" table to match those
               included in the "Unbundled Network Elements" report.

  Modify the code in ICM as follows:

   - Rename the column description, included in the Expense Module base table,
     from "Other Direct" to "Direct Other".

   - Rename the "Other Direct" field contained throughout the Expense Module
     source code of ICM to "Direct Other".

   - Rename the column heading found in the Expense Results table as
     "Capital Rec" to "Depreciation and Return".  This change will result in a
     consistent naming convention between the two reports.

   - Rename the column heading found in the Expense Results table as
     "Composite Tax" to "Composite Income Tax".  This change will result in a
     consistent naming convention between the two reports.

   - Rename the column heading found in the Expense Results table as "Pool Factor"
     to "Maintenance and Support".  This change will result in a consistent
     naming convention between the two reports.
}

{ RFPC 0039  - Expand the list of all valid USOA accounts in the ICM source code
               applicable for the calculation of Common Costs for all states.
```

```
37     Originated 3/23/99
38
39     This change will accommodate the correct reporting of common costs for all
40     applicable accounts for each state.
41
42     Description of the program change desired:
43
44     To eliminate the need for continuous updates to the ICM code related to
45     additional USOA accounts required for the common cost calculation of individual
46     states, please modify the code to include all valid USOA accounts in the
47     computation of common costs.
48   }
49
50   { RFPC 0038  - Expand the list of valid USOA accounts in the ICM source code
51                  applicable for the calculation of Common Costs for South Carolina.
52     Originated 3/23/99
53
54     This change will accommodate the correct reporting of common costs for all
55     applicable accounts for the state of South Carolina.
56
57     Description of the program change desired:
58
59     In order to correct the calculation and reporting of common costs for South
60     Carolina within ICM,  please modify the ICM source code to include the USOA
61     accounts listed.
62
63     USOA  Acct            Account Description          Retail / Wholesale
64
65     2115                  Garage Work Equipment          Retail
66     2115                  Garage Work Equipment          Wholesale
67     6623                  Customer Services       Wholesale
68
69   }
70
71   { RFPC 0037  - Rename all occurrences of the field "Transport", included in the
72                  Expense Module ICM source code and the Expense Module base table
73                  column description, to "Transmission".
74     Originated 4/2/99
75
```

```
   The request chnage will provide a more accurate depiction of the functional
   activity.

   Description of the program change desired:

   · Rename the column description, included in the Expense Module base table,
     from "Transport" to "Transmission".

   · Rename the "Transport" field contained throughout the Expense Module source
     code of ICM to "Transmission".
}

{ RFPC 0034  - Depreciation and Return and Composite Income Tax Calculation revision
               for Non-Depreciable Assets
  Originated 03/01/99

  This change will correct the calculation and presentation of the depreciation
  component of the Depreciation and Return factor and composite income tax factor
  contained in ICM for non-depreciable assets.

  Description of the program change desired:

  In order to correctly present and calculate the Depreciation and Return factor and
  composite income tax factor within ICM, the following modifications to the
  ICM code need to be made:

  · Allow for the inclusion of zero (0) years as a valid value for depreciable
    lives in ICM

  · Revise the ICM code for the calculation of the Depreciation and Return factor and
    composite income tax rate to eliminate the division by zero problem.
}

{ RFPC 0033  - Correct the shared cost assignment used in the calculation of
               Direct Other Shared Costs
  Originated 03/01/99

  This change will result in the consistent assignment of Direct Other costs in
  ICM regardless of whether the costs are included as part of the UNE
```

```
   expense / investment ratio (including shared costs) or separately as a
   component of the Network Services Shared Costs.

   Description of the program change desired:

   Direct Other Costs are identified as Network administration and operations
   costs.  These costs are assigned to the Network Cost Pools comprised of Cable,
   Aerial Fiber, Aerial Copper, Buried Fiber, Buried Copper, Underground Fiber,
   Underground Copper, Poles, Conduit, Transmission, Switch and IOT.

   Revise the assignment of Direct Other costs, for purposes of calculating the
   Network Services Shared Cost, from the procedure listed above, to an
   assignment based on the relationship between the sum of direct and shared
   costs for each network cost pool and the sum of the direct and shared costs
   for all Network cost pools.  This methodology is consistent with the
   assignment used to develop total cost pool expenses for the purpose of
   computing the expense/investment ratios (including shared costs).
}

{ RFPC 0025  - Investment Cable Cost Pool Assignment Revision
  Originated 01/21/99

  The request for change outlined above is necessary to ensure that the cost pool
  assignments, by account, are consistent for both expenses and investments.

  Description of the program change desired:

  Revise the investment cost pool assignment, hard coded in ICM, for the below
  listed investment accounts from the "Cable" cost pool to the specified cable
  cost pools.

                                          Original    Revised
  Account   Acct Description              Cost Pool   Cost Pool

  2424.10   Submarine Cable - Metallic        Cable       Buried Metallic
  2424.20   Submarine Cable - Fiber             Cable           Buried Non-Metallic

  2425.10   Deep Sea Cable - Metallic         Cable       Buried Metallic
  2425.20   Deep Sea Cable - Fiber              Cable           Buried Non-Metallic
```

```
154
155     2426.10   Intra-bldg Ntwk Cable - Metallic    Cable       Aerial Metallic
156     2426.20   Intra-bldg Ntwk Cable - Fiber         Cable           Aerial Non-Metallic
157   }
158
159   { RFPC 0006  -  Joint / Shared Cost
160     Originated 9/28/98
161
162     Program change will allow ICM to be in compliance with the Michigan, Texas,
163     and OANAD cost rules/order requiring that costs be expressed in the requested
164     format.
165
166     Description of the program change desired:
167
168     - Develop methodology to identify shared cost by: BNF category and service category.
169     - Investigate current ICM "Volume Insensitive cost" so that costs are BNF, BNF
170       category, service or service category specific.
171     - Identify common cost separately from shared BNF category and shared service.
172     - Compare methodology to that submitted under Rule 23.91 (TX).
173       (note: service means retail, wholesale, or UNE service)
174     - Program change to discretely identify joint/shared cost by BNF category and
175       service category. It would be beneficial if the user is able to control the
176       assignment of shared, volume insensitive costs to: a particular BNF, a
177       particular BNF category, a particular service (e.g. 2W loop UNE), a particular
178       service category.
179     - Output report listing direct cost, joint/shared cost, TSLRIC/TLRIC cost and
180       common cost by BNF, BNF category, service and service category.
181     - Two new tables will be created, ExpCCWorkSheet (Common Cost Worksheet) and
182       ExpJointShared (Joint/Shared Costs).
183     - Add two new fields to the expense base table (Total Revenues - type float,
184       and Data Year - type integer)
185   }
186
187   { RFPC 0004  -  Distribution of cable investment among other cable cost pools
188     Originated 9/23/98
189
190     This will match the mapping process that is done for the related expenses.
191
192     Description of the program change desired:
```

```
193     · This change is intended to parallel the assignment that occurs to the cable
194       expense cost pool.
195     · Submarine Metallic (242410), Submarine Non-Metallic (242420),Deep Sea Metallic
196       (242510), Deep Sea Non-Metallic (242520), Intrabuilding Metallic (242610),
197        and Intrabuilding Non-Metallic (242620) are all currently mapped to the
198        cable cost pool.  These amounts should be assigned to the other cable cost
199        pools, ie. Aerial Metallic (242110), Aerial Non-Metallic (242120), Buried
200        Metallic (242210), Buried Non-Metallic (242220), Underground Metallic
201        (242310), and Underground Non-Metallic (242320).
202     · This would be done by summing the amounts in Submarine Metallic, Submarine
203       Non-Metallic, Deep Sea Metallic, Deep Sea Non-Metallic, Intrabuilding Metallic,
204       and Intrabuilding Non-Metallic into a "Unmapped Cable Pool".
205     · The remaining other cable cost pools, Aerial Metallic, Aerial Non-Metallic,
206       Underground Metallic, Underground Non-Metallic, Buried Metallic, and Buried
207       Non-Metallic, should be summed then each individual cost pool divided by the
208       total to determine each "other" cable cost pool percentage of the total.
209     · The percentage for each other cable cost pool would then be multiplied by
210       the "Unmapped Cable Pool" total to compute the amount of cable investment
211       assigned to each "other" cable cost pool.
212     · This amount would then be added to the amount of that cost pool and totaled.
213     · This assignment will eliminate any amounts recorded in the Cable Investment Cost Pool.
214   }
215   interface
216
217   uses
218     SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
219     Forms, Dialogs, StdCtrls, ExtCtrls, DBCtrls, Grids, DBGrids, Menus, DB,
220     DBTables, Printers, Assoc, Tabs, Globalda, TablePrnt, Import, Gen_Exp, Monitr;
221
222   type
223     Tf_Expense = class(TForm)
224       l_CopyRight: TLabel;       // copyright label displayed on screen
225       t_Expense: TTable;         // Expense investment table (3002)
226       ds_Expense: TDataSource;
227       sd_Expense: TSaveDialog;
228       od_Expense: TOpenDialog;
229       MainMenu1: TMainMenu;
230       File1: TMenuItem;
231       New1: TMenuItem;
```

```
232       Open1: TMenuItem;
233       N1: TMenuItem;
234       Import1: TMenuItem;
235       Export1: TMenuItem;
236       N2: TMenuItem;
237       Exit1: TMenuItem;
238       Help1: TMenuItem;
239       dbg_Expense: TDBGrid;      // expense database grid
240       dbn_Expense: TDBNavigator;  // expense database navigator
241       cb_Assoc: TCheckBox;       // Update Association checkbox
242       Print1: TMenuItem;
243       N3: TMenuItem;
244       PrintDialog1: TPrintDialog;
245       ts_expense: TTabSet;
246       t_ExpBase: TTable;         // Expense Base Table (3006)
247       t_ExpLife: TTable;           // Plant Life Table (3005)
248       t_ExpResult: TTable;         // Expense Result Table (created)
249       t_roexpense: TTable;         // Expense Run Time Options Table
250       t_PropTax: TTable;
251       t_JointSharedCost: TTable;
252       t_Worksheets: TTable;
253       procedure FormResize(Sender: TObject);  // resizes form
254       procedure FormShow(Sender: TObject);    // displays form
255       procedure Exit1Click(Sender: TObject);  // responds to Exit selection from menu
256       procedure FormHide(Sender: TObject);    // makes form invisible
257       procedure New1Click(Sender: TObject);   // responds to New selection from menu
258       procedure Open1Click(Sender: TObject);  // responds to Open selection from menu
259       procedure Import1Click(Sender: TObject);// responds to Import selection from menu
260       procedure Export1Click(Sender: TObject);// responds to Export selection from menu
261       procedure Print1Click(Sender: TObject); // responds to Print selection from menu
262       procedure ts_expenseChange(Sender: TObject; NewTab: Integer;
263         var AllowChange: Boolean);            // change to selected tab on window
264       procedure Help1Click(Sender: TObject);  // responds to Help selection from menu
265       procedure FormActivate(Sender: TObject);
266     private
267       { Private declarations }
268       procedure SetCaption;     // defines the caption on the form (includes the table name)
269       procedure Make_Expense;     // display/create the Expense table
270       procedure Make_ExpBase;     // display/create the Expense Base table
```

```
271       procedure Make_ExpLife;      // display/create the Plant Life table
272       procedure Make_PropTax;      // display/create the Plant Life table
273       procedure Make_JointShared; // create the Joint/Shared Cost table and open for output (RFPC 0006)
274       procedure Make_Worksheets;  // create the Common Cost Worksheet table and open for output (RFPC 0006)
275       function  PV(amount,rate, year: double) : double;
276       function  PMT(amount,rate,year: double): double;
277       function  SFD(salvage,rate,year: double): double;
278       function  DepreciationAndReturn (rate_return, life, salvage : Double): Double;  //RFPC0051
279       function  CompositeIncomeTax(rate_return, life, salvage : Double; macrs_yr : integer): Double;
280       function  PV_Macrs (macrs_yr : integer; rate : double): double;
281       procedure DoTabChange(NewTab: Integer);
282       procedure UpdateJointSharedCosts(Market,ActNo: string;  Value:double); // RFPC 0006
283       function  GetActDesc(Act:string):string;   //RFPC 0006
284       procedure DirectExpAndRatios(Market: string);
285       procedure CommonCostWorksheets(Market: string);
286     public
287       { Public declarations }
288       procedure Entry;
289       procedure PrintExpense( Open : Boolean);
290       procedure PrintExpLife( Open : Boolean);
291       procedure PrintExpBase( Open : Boolean);
292       procedure PrintPropTax( Open : Boolean);
293       procedure CalcFactors;
294       procedure Report;
295       procedure NetworkServicesReport;
296       procedure SummaryReport;
297     end;
298
299   var
300     f_Expense         : Tf_Expense;
301     TabName           : String;  // name of tab selected (Account / Plant Life / Base)
302     Exp_Cable         : Double = 0;         // Cable expense
303     Exp_FiAer         : Double = 0;         // Aerial Fiber expense
304     Exp_CuAer         : Double = 0;         // Aeriel Copper expense
305     Exp_FiBur         : Double = 0;         // Buried Fiber expense
306     Exp_CuBur         : Double = 0;         // Buried Copper expense
307     Exp_FiUnd         : Double = 0;         // Underground Fiber expense
308     Exp_CuUnd         : Double = 0;         // Underground Copper expense
309     Exp_Poles         : Double = 0;         // Poles expense
```

```
310     Exp_Conduit      : Double = 0;        // Conduit expense
311     Exp_Transmission : Double = 0;        // Transmission expense
312     Exp_Switch       : Double = 0;        // Switch expense
313     Exp_Access       : Double = 0;        // Access expense
314     Exp_IOT          : Double = 0;        // Information, Originating & Terminating expense
315     Exp_OD           : Double = 0;        // Direct Other expense
316     Exp_BillPay      : Double = 0;        // Billing & Collection expense
317     Exp_OperSvc      : Double = 0;        // Operator Services expense
318     Exp_PubCom       : Double = 0;        // Public Communications expense
319     Exp_Consumer     : Double = 0;        // Consumer expense
320     Exp_Business     : Double = 0;        // Business expense
321     Exp_IXC          : Double = 0;        // IXC expense
322     Exp_Common       : Double = 0;        // Common expense
323     Inv_Cable        : Double = 0;        // Investment for Total Cable (amount * adjust factors)
324     Inv_FiAer        : Double = 0;        // Investment for Aerial Fiber (amount * adjust factors)
325     Inv_CuAer        : Double = 0;        // Investment for Aerial Copper (amount * adjust factors)
326     Inv_FiBur        : Double = 0;        // Investment for Buried Fiber (amount * adjust factors)
327     Inv_CuBur        : Double = 0;        // Investment for Buried Copper (amount * adjust factors)
328     Inv_FiUnd        : Double = 0;        // Investment for Underground Fiber (amount * adjust factors)
329     Inv_CuUnd        : Double = 0;        // Investment for Underground Copper (amount * adjust factors)
330     Inv_Poles        : Double = 0;        // Investment for Poles (amount * adjust factors)
331     Inv_Conduit      : Double = 0;        // Investment for Conduit (amount * adjust factors)
332     Inv_Transmission : Double = 0;        // Investment for Transmission (amount * adjust factors)
333     Inv_Switch       : Double = 0;        // Investment for Switch (amount * adjust factors)
334
335 implementation
336    Uses Nemain, Repview, Calibrat;
337 {$R *.DFM}
338
339 type
340   ArrayDouble = array [0 .. 100] of double;
341   RecType = record
342               Act    : string[6];
343               Factor : double;
344               Total  : double;
345             end;
346   RecArray = array [1 .. 100] of RecType;
347 var
348   SubTotal, DirectExp, TotInv, RedistributeTot,
```

```
349     Ratios : ArrayDouble;   //RFPC 0006
350     GrandTotal : array [1 .. 5] of double;              //RFPC 0006
351     TotalIOT : double;                                  //RFPC 0006
352     DataYear  : word;                                   //RFPC 0006
353     TotRev    : double;                                 //RFPC 0006
354     Market, Category   : String;                        //RFPC 0006
355     ExpArray : RecArray;                                // RFPC 0006 Expense accounts and totals
356     InvArray : RecArray;                                // RFPC 0006 Investment accounts and totals
357     NumExpActs, NumInvActs : integer;                   // RFPC 0006
358     SubTotalCable_IOT : double;                         // RFPC 0033
359     AccountStr : string;                                // RFPC 0034 - account number for error msgs
360
361   procedure Tf_Expense.Entry;
362   begin
363     Show;
364   end;
365
366   // RFPC 0006
367   // Update the joint/shared costs with the direct expense totals, subtotals and
368   // ratios
369   procedure Tf_Expense.DirectExpAndRatios(Market: string);
370   var i : integer;
371       TempArray : ArrayDouble;
372       totNonNetwork, totShared, TempTot : double;
373    procedure RemoveTempRecs(Accounts : array of string);
374     var j : integer;
375     begin
376       with t_JointSharedCost do
377         begin
378           CancelRange;
379           First;
380           // remove temp records
381           for j:=low(Accounts) to High(Accounts) do
382             if FindKey([Market,Copy(Accounts[j],2,10)]) then
383               Delete;
384
385           // remove the "Z"s from the accounts
386           for j:=low(Accounts) to High(Accounts) do
387             if FindKey([Market,Accounts[j]]) then
```

```
388               begin
389                 Edit;
390                 FieldByName('Account').AsString:=
391                   Copy(FieldByName('Account').AsString,2,12);
392                 Post;
393               end;
394         end;
395     end;
396   begin
397     // initialize the arrays
398     FillChar(SubTotal,SizeOf(ArrayDouble),0);
399     FillChar(DirectExp,SizeOf(ArrayDouble),0);
400     FillChar(Ratios,SizeOf(ArrayDouble),0);
401     FillChar(TotInv,SizeOf(ArrayDouble),0);
402     // set the range and cycle through the table computing the totals
403     with t_JointSharedCost do
404       begin
405         CancelRange;
406         if FindKey([Market,'C) Total Investment']) then
407           for i:=3 to 13 do TotInv[i]:=Fields[i].AsFloat;
408         SetRange([Market,'6112'],[Market,'7240']);
409         First;
410         while not eof do
411           begin
412             FillChar(TempArray,SizeOf(ArrayDouble),0);
413             for i:=2 to 23 do TempArray[i]:=Fields[i].AsFloat;
414             // compute the subtotals
415             for i:=2 to 23 do SubTotal[i]:=SubTotal[i]+TempArray[i];
416             // compute the direct expenses
417             if (FieldByName('Account').AsString >= '6211') and (FieldByName('Account').AsString <= '6441')
418   then
419               for i:=3 to 15 do
420                 begin
421                   DirectExp[i]:=DirectExp[i]+TempArray[i];
422                   DirectExp[2]:=DirectExp[2]+TempArray[i];
423                 end;
424             Next;   // get the next record
425           end; {while not eof}
426
```

```
427         // RFPC 0033
428         // calculate the sum of Cable thru IOT for SubTotal
429         SubTotalCable_IOT:=0;
430         for i:=3 to 14 do SubTotalCable_IOT:=SubTotalCable_IOT+SubTotal[i];
431
432         // round all the information currently in the table
433         // first cancel the range and go to the top of the table
434         CancelRange;
435         if not BOF then First;
436         while not eof do
437           begin
438             Edit; // put it in edit mode
439             // round all the numbers to the nearest dollar
440             for i:=2 to 23 do
441               if (Fields[i].AsString <> '') then Fields[i].AsFloat:=Round(Fields[i].AsFloat);
442             Post;   // update the record
443             Next;   // get the next record
444           end; {while not eof}
445
446         // removes all temporary records in the table, per Joe Abs
447         RemoveTempRecs(['Z6112','Z6113','Z6114','Z6115','Z6116','Z6121','Z6122','Z6123','Z6124']);
448
449         // first cancel the range and go to the top of the table
450         CancelRange;
451         if not BOF then First;
452
453         // save out the state of the flag in the table
454         Append;
455         if ExpShared then FieldByName('Account').AsString:='1-Included'
456         else FieldByName('Account').AsString:='0-Excluded';
457         FieldByName('Category').AsString:=Market;
458         Post;
459
460         // save out blank lines
461         Append;
462         FieldByName('Category').AsString:=Market;
463         FieldByName('Account').AsString:='E)';
464         Post;
465         Append;
```

```
466         FieldByName('Category').AsString:=Market;
467         FieldByName('Account').AsString:='F) Expense Summary:';
468         Post;
469
470         // save out the subtotals
471         Append;      // append the record
472         FieldByName('Category').AsString:=Market;
473         FieldByName('Account').AsString:='A) Subtotal';
474         for i:=2 to 23 do Fields[i].AsFloat:=Round(SubTotal[i]);
475         Post;        // post the record
476
477         // save out the DirectExp totals
478         Append;      // append the record
479         FieldByName('Category').AsString:=Market;
480         FieldByName('Account').AsString:='G) Direct Cost';
481         for i:=2 to 14 do Fields[i].AsFloat:=Round(DirectExp[i]);
482         Post;        // post the record
483         if not ExpShared then
484           begin
485             Append;      // append the record
486             FieldByName('Category').AsString:=Market;
487             FieldByName('Account').AsString:='B) Direct Cost';
488             for i:=2 to 14 do Fields[i].AsFloat:=Round(DirectExp[i]);
489             Post;        // post the record
490           end;
491
492
493         // save out the Redistributed totals
494         if ExpShared then
495           begin
496             Temptot:=0;
497             Append;      // append the record
498             FieldByName('Category').AsString:=Market;
499             FieldByName('Account').AsString:='B) Total after redistribution';
500             for i:=4 to 14 do
501               begin
502                 Fields[i].AsFloat:=Round(RedistributeTot[i]);
503                 TempTot:=TempTot+RedistributeTot[i];
504               end;
```

```
505             for i:=16 to 23 do
506               begin
507                 Fields[i].AsFloat:=Round(Subtotal[i]);
508                 TempTot:=TempTot+Subtotal[i];
509               end;
510             FieldByName('Total').AsFloat:=Round(TempTot);
511             Post;       // post the record
512           end;
513
514         // save out the Ratios totals
515         for i:=3 to 13 do
516           if ExpShared and (TotInv[i] <> 0) and (RedistributeTot[i] <> 0) then
517             Ratios[i]:=RedistributeTot[i]/TotInv[i]
518           else if not ExpShared and (TotInv[i] <> 0) and (DirectExp[i] <> 0) then
519             Ratios[i]:=DirectExp[i]/TotInv[i]
520           else Ratios[i]:=0;
521         Append;     // append the record
522         FieldByName('Category').AsString:=Market;
523         FieldByName('Account').AsString:='D) Ratios (B/C)';
524         for i:=3 to 13 do Fields[i].AsFloat:=Ratios[i];
525         Post;       // post the record
526
527         // save out the Non-Network Cost
528         Append;
529         totNonNetwork:=(SubTotal[16]+SubTotal[17]+SubTotal[18]+SubTotal[19]+
530                         SubTotal[20]+SubTotal[21]+SubTotal[22]+SubTotal[23]);
531         if (totNonNetwork < 0) then totNonNetwork:=0;
532         FieldByName('Account').AsString:='I) Non-Network Cost';
533         FieldByName('Category').AsString:=Market;
534         FieldByName('Total').AsFloat:=Round(totNonNetwork);
535         Post;
536
537         // save out the Shared total
538         Append;
539         totShared:=(SubTotal[2]-(DirectExp[2]+totNonNetwork));
540         if (totNonNetwork < 0) then totNonNetwork:=0;
541         FieldByName('Account').AsString:='H) Shared Cost';
542         FieldByName('Category').AsString:=Market;
543         FieldByName('Total').AsFloat:=Round(totShared);
```

```
544         Post;
545
546         // save out the Summary Total
547         Append;
548         FieldByName('Account').AsString:='I) Summary Total';
549         FieldByName('Category').AsString:=Market;
550         FieldByName('Total').AsFloat:=Round(SubTotal[2]);
551         Post;
552
553         Close;      // close the table
554       end; {with t_JointSharedCost}
555   end; {DirectExpAndRatios}
556
557   // RFPC 0006
558   // Update the joint/shared costs with the direct expense totals, subtotals and
559   // ratios
560   procedure Tf_Expense.CommonCostWorksheets(Market: string);
561   var i, j : integer;
562    procedure SaveExpInv(Accounts : array of string;
563                         var NumActs: integer;
564                         var ActArray: RecArray);
565     var NewAmt, Total : double;
566         i : integer;
567     begin
568       // expense table is already open. save out the information provided that
569       // the account number exists in the table
570       with t_Expense do
571         begin
572           if not BOF then First;  // put at the top of the table
573           for i := Low(Accounts) to High(Accounts) do
574             if FindKey([Accounts[i]+'00',Market]) then
575               begin
576                 Inc(NumActs);
577                 ActArray[NumActs].Act:=Accounts[i];
578                 ActArray[NumActs].Total:=0;
579                 t_Worksheets.Append;
580                 t_Worksheets.FieldByName('Account').AsString:=Accounts[i];
581                 t_Worksheets.FieldByName('Category').AsString:=Market;
582                 t_Worksheets.FieldByName('Amount').AsFloat:=FieldByName('Amount').AsFloat;
```

```
583                t_Worksheets.FieldByName('Adjust1').AsFloat:=FieldByName('Adjust1').AsFloat;
584                t_Worksheets.FieldByName('Adjust2').AsFloat:=FieldByName('Adjust2').AsFloat;
585                t_Worksheets.FieldByName('Adjust3').AsFloat:=FieldByName('Adjust3').AsFloat;
586                NewAmt := FieldByName('Amount').AsFloat*
587                 (1.0 - (1.0 - FieldByName('Adjust1').AsFloat) - (1.0 - FieldByName('Adjust2').AsFloat))*
588                 FieldByName('Adjust3').AsFloat;
589                t_Worksheets.FieldByName('New Amount').AsFloat:=NewAmt;
590                t_Worksheets.FieldByName('%BC').AsFloat:=FieldByName('Bill_Pay').AsFloat;
591                if (UpperCase(Market)='R') then
592                  t_Worksheets.FieldByName('%Consumer').AsFloat:=FieldByName('Consumer').AsFloat;
593                if (UpperCase(Market)='R') then
594                  t_Worksheets.FieldByName('%Business').AsFloat:=FieldByName('Business').AsFloat;
595                t_Worksheets.FieldByName('%CXR').AsFloat:=FieldByName('CXR').AsFloat;
596                t_Worksheets.FieldByName('%Common').AsFloat:=FieldByName('Common').AsFloat;
597                t_Worksheets.FieldByName('$BC').AsFloat:=NewAmt*FieldByName('Bill_Pay').AsFloat;
598                if (UpperCase(Market)='R') then
599                  t_Worksheets.FieldByName('$Consumer').AsFloat:=NewAmt*FieldByName('Consumer').AsFloat;
600                if (UpperCase(Market)='R') then
601                  t_Worksheets.FieldByName('$Business').AsFloat:=NewAmt*FieldByName('Business').AsFloat;
602                t_Worksheets.FieldByName('$CXR').AsFloat:=NewAmt*FieldByName('CXR').AsFloat;
603                t_Worksheets.FieldByName('$Common').AsFloat:=NewAmt*FieldByName('Common').AsFloat;
604                if (UpperCase(Market)='R') then
605                  Total:=t_Worksheets.FieldByName('$BC').AsFloat+
606                         t_Worksheets.FieldByName('$Consumer').AsFloat+
607                         t_Worksheets.FieldByName('$Business').AsFloat+
608                         t_Worksheets.FieldByName('$CXR').AsFloat+
609                         t_Worksheets.FieldByName('$Common').AsFloat
610                else
611                  Total:=t_Worksheets.FieldByName('$BC').AsFloat+
612                         t_Worksheets.FieldByName('$CXR').AsFloat+
613                         t_Worksheets.FieldByName('$Common').AsFloat;
614                // get the Depreciation and Return, Composite Income Tax, and Property tax from the result
615   table
616                if (Accounts[i][1]='2') and t_ExpResult.FindKey([Accounts[i]+'00']) then
617                  begin
618                    ActArray[NumActs].Factor:=t_ExpResult.FieldByName('Depreciation and Return').AsFloat+
619                                              t_ExpResult.FieldByName('Composite Income Tax').AsFloat+
620                                              t_ExpResult.FieldByName('Property Tax').AsFloat;
621                    t_Worksheets.FieldByName('ICM Based Factors').AsFloat:=ActArray[NumActs].Factor;
```

```
                    t_Worksheets.FieldByName('Subtotal').AsFloat:=Round(Total);
                    t_Worksheets.FieldByName('Total').AsFloat:=Round(Total*ActArray[NumActs].Factor);
                    Total:=(Total*ActArray[NumActs].Factor);
                  end
                else t_Worksheets.FieldByName('Total').AsFloat:=Round(Total);
                t_Worksheets.Post;
                ActArray[NumActs].Total:=Total;
              end; {if key found}
          Next;
        end; {with}
    end;
begin
  // blank out the expense array
  for i:=low(ExpArray) to High(ExpArray) do
    begin
      ExpArray[i].Act:='';
      ExpArray[i].Factor:=0;
      ExpArray[i].Total:=0;
    end; {for exparray loop}
  // blank out the investment array
  for i:=low(InvArray) to High(InvArray) do
    begin
      InvArray[i].Act:='';
      InvArray[i].Factor:=0;
      InvArray[i].Total:=0;
    end; {for invarray loop}
  NumExpActs:=0;   NumInvActs:=0;

  // save out the expense info
  //RFPC 0038 and 0039
  SaveExpInv(['6112','6113','6114','6115','6116','6121','6122','6123','6124',
              '6211','6212','6215','6220','6231','6232','6311','6341','6351','6362',
              '6411','6421','6422','6423','6424','6425','6426','6431','6441',
              '6512','6531','6532','6533','6534','6535',
              '6611','6612','6613','6621','6622','6623',
              '6711','6712','6721','6722','6723','6724','6725','6726','6727','6728','7240'],
             NumExpActs,ExpArray);
  // save out the investment info
  SaveExpInv(['2111','2112','2113','2114','2115','2116','2121','2122','2123','2124'],NumInvActs,InvArray);
```

```
661
662     t_Worksheets.Close;  // close the worksheets result table
663   end; {CommonCostWorksheets}
664
665   // RFPC 0006
666   // Update the joint/shared costs for the specified market and account number.
667   // If a record, market and account, is found then add the new cost to the
668   // existing ones, else append a new record.
669   procedure Tf_Expense.UpdateJointSharedCosts(Market,ActNo: string;  Value:double);
670   begin
671     with t_JointSharedCost do
672       begin
673         if FindKey([Market,ActNo]) then
674           begin  // add the information into the existing record
675             // put it in edit mode
676             Edit;
677             if (Value <> 0.0) then
678               FieldByName('Total').AsFloat := FieldByName('Total').AsFloat + Value;
679             if (Value <> 0.0) and (t_Expense.FieldByName('Cable').AsFloat <> 0.0) then
680               FieldByName('Cable').AsFloat := FieldByName('Cable').AsFloat +
681                   (Value * t_Expense.FieldByName('Cable').AsFloat);
682             if (Value <> 0.0) and (t_Expense.FieldByName('FiAer').AsFloat <> 0.0) then
683               FieldByName('Aerial Non-Metallic').AsFloat := FieldByName('Aerial Non-Metallic').AsFloat +
684                   (Value * t_Expense.FieldByName('FiAer').AsFloat);
685             if (Value <> 0.0) and (t_Expense.FieldByName('FiBur').AsFloat <> 0.0) then
686               FieldByName('Buried Non-Metallic').AsFloat := FieldByName('Buried Non-Metallic').AsFloat +
687                   (Value * t_Expense.FieldByName('FiBur').AsFloat);
688             if (Value <> 0.0) and (t_Expense.FieldByName('FiUnd').AsFloat <> 0.0) then
689               FieldByName('Underground Non-Metallic').AsFloat := FieldByName('Underground Non-
690   Metallic').AsFloat +
691                   (Value * t_Expense.FieldByName('FiUnd').AsFloat);
692             if (Value <> 0.0) and (t_Expense.FieldByName('CuAer').AsFloat <> 0.0) then
693               FieldByName('Aerial Metallic').AsFloat := FieldByName('Aerial Metallic').AsFloat +
694                   (Value * t_Expense.FieldByName('CuAer').AsFloat);
695             if (Value <> 0.0) and (t_Expense.FieldByName('CuBur').AsFloat <> 0.0) then
696               FieldByName('Buried Metallic').AsFloat := FieldByName('Buried Metallic').AsFloat +
697                   (Value * t_Expense.FieldByName('CuBur').AsFloat);
698             if (Value <> 0.0) and (t_Expense.FieldByName('CuUnd').AsFloat <> 0.0) then
699               FieldByName('Underground Metallic').AsFloat := FieldByName('Underground Metallic').AsFloat +
```

```
700             (Value * t_Expense.FieldByName('CuUnd').AsFloat);
701           if (Value <> 0.0) and (t_Expense.FieldByName('Poles').AsFloat <> 0.0) then
702             FieldByName('Poles').AsFloat := FieldByName('Poles').AsFloat +
703             (Value * t_Expense.FieldByName('Poles').AsFloat);
704           if (Value <> 0.0) and (t_Expense.FieldByName('Conduit').AsFloat <> 0.0) then
705             FieldByName('Conduit').AsFloat := FieldByName('Conduit').AsFloat +
706             (Value * t_Expense.FieldByName('Conduit').AsFloat);
707           // RFPC 0037
708           if (Value <> 0.0) and (t_Expense.FieldByName('Transmission').AsFloat <> 0.0) then
709             FieldByName('Transmission').AsFloat := FieldByName('Transmission').AsFloat +
710             (Value * t_Expense.FieldByName('Transmission').AsFloat);
711           if (Value <> 0.0) and (t_Expense.FieldByName('Switch').AsFloat <> 0.0) then
712             FieldByName('Switch').AsFloat := FieldByName('Switch').AsFloat +
713             (Value * t_Expense.FieldByName('Switch').AsFloat);
714           if (Value <> 0.0) and (t_Expense.FieldByName('Access').AsFloat <> 0.0) then
715             FieldByName('Access').AsFloat := FieldByName('Access').AsFloat +
716             (Value * t_Expense.FieldByName('Access').AsFloat);
717           if (Value <> 0.0) and (t_Expense.FieldByName('IOT').AsFloat <> 0.0) then
718             FieldByName('IOT').AsFloat := FieldByName('IOT').AsFloat +
719             (Value * t_Expense.FieldByName('IOT').AsFloat);
720           if (Value <> 0.0) and (t_Expense.FieldByName('Direct Other').AsFloat <> 0.0) then
721             FieldByName('Direct Other').AsFloat := FieldByName('Direct Other').AsFloat +
722             (Value * t_Expense.FieldByName('Direct Other').AsFloat);
723           if (Value <> 0.0) and (t_Expense.FieldByName('Bill_Pay').AsFloat <> 0.0) then
724             FieldByName('B and C').AsFloat := FieldByName('B and C').AsFloat +
725             (Value * t_Expense.FieldByName('Bill_Pay').AsFloat);
726           if (Value <> 0.0) and (t_Expense.FieldByName('Oper_Serv').AsFloat <> 0.0) then
727             FieldByName('Oper').AsFloat := FieldByName('Oper').AsFloat +
728             (Value • t_Expense.FieldByName('Oper_Serv').AsFloat);
729           if (Value <> 0.0) and (t_Expense.FieldByName('PubCom').AsFloat <> 0.0) then
730             FieldByName('PUB COMM').AsFloat := FieldByName('PUB COMM').AsFloat +
731             (Value * t_Expense.FieldByName('PubCom').AsFloat);
732           if (Value <> 0.0) and (t_Expense.FieldByName('Consumer').AsFloat <> 0.0) then
733             FieldByName('Consumer').AsFloat := FieldByName('Consumer').AsFloat +
734             (Value * t_Expense.FieldByName('Consumer').AsFloat);
735           if (Value <> 0.0) and (t_Expense.FieldByName('Business').AsFloat <> 0.0) then
736             FieldByName('Business').AsFloat := FieldByName('Business').AsFloat +
737             (Value * t_Expense.FieldByName('Business').AsFloat);
738           if (Value <> 0.0) and (t_Expense.FieldByName('CXR').AsFloat <> 0.0) then
```

```
739             FieldByName('CXR').AsFloat := FieldByName('CXR').AsFloat +
740                (Value * t_Expense.FieldByName('CXR').AsFloat);
741           if (Value <> 0.0) and (t_Expense.FieldByName('Common').AsFloat <> 0.0) then
742             FieldByName('Common').AsFloat := FieldByName('Common').AsFloat +
743                (Value * t_Expense.FieldByName('Common').AsFloat);
744         end
745       else
746         begin  // append the new information
747           Append;
748           FieldByName('Account').AsString     := ActNo;
749           FieldByName('Category').AsString    := Market;
750           if (Value = 0.0) then FieldByName('Total').AsFloat :=0.0
751           else FieldByName('Total').AsFloat := Value;
752           if (Value = 0.0) or (t_Expense.FieldByName('Cable').AsFloat=0.0) then
753             FieldByName('Cable').AsFloat := 0.0
754           else FieldByName('Cable').AsFloat := (Value * t_Expense.FieldByName('Cable').AsFloat);
755           if (Value = 0.0) or (t_Expense.FieldByName('FiAer').AsFloat=0.0) then
756             FieldByName('Aerial Non-Metallic').AsFloat := 0.0
757           else FieldByName('Aerial Non-Metallic').AsFloat := (Value *
758 t_Expense.FieldByName('FiAer').AsFloat);
759           if (Value = 0.0) or (t_Expense.FieldByName('FiBur').AsFloat=0.0) then
760             FieldByName('Buried Non-Metallic').AsFloat := 0.0
761           else FieldByName('Buried Non-Metallic').AsFloat := (Value *
762 t_Expense.FieldByName('FiBur').AsFloat);
763           if (Value = 0.0) or (t_Expense.FieldByName('FiUnd').AsFloat=0.0) then
764             FieldByName('Underground Non-Metallic').AsFloat := 0.0
765           else FieldByName('Underground Non-Metallic').AsFloat := (Value *
766 t_Expense.FieldByName('FiUnd').AsFloat);
767           if (Value = 0.0) or (t_Expense.FieldByName('CuAer').AsFloat=0.0) then
768             FieldByName('Aerial Metallic').AsFloat := 0.0
769           else FieldByName('Aerial Metallic').AsFloat := (Value * t_Expense.FieldByName('CuAer').AsFloat);
770           if (Value = 0.0) or (t_Expense.FieldByName('CuBur').AsFloat=0.0) then
771             FieldByName('Buried Metallic').AsFloat := 0.0
772           else FieldByName('Buried Metallic').AsFloat := (Value * t_Expense.FieldByName('CuBur').AsFloat);
773           if (Value = 0.0) or (t_Expense.FieldByName('CuUnd').AsFloat=0.0) then
774             FieldByName('Underground Metallic').AsFloat := 0.0
775           else FieldByName('Underground Metallic').AsFloat := (Value *
776 t_Expense.FieldByName('CuUnd').AsFloat);
777           if (Value = 0.0) or (t_Expense.FieldByName('Poles').AsFloat=0.0) then
```

```
778             FieldByName('Poles').AsFloat := 0.0
779           else FieldByName('Poles').AsFloat := (Value * t_Expense.FieldByName('Poles').AsFloat);
780           if (Value = 0.0) or (t_Expense.FieldByName('Conduit').AsFloat=0.0) then
781             FieldByName('Conduit').AsFloat := 0.0
782           else FieldByName('Conduit').AsFloat := (Value * t_Expense.FieldByName('Conduit').AsFloat);
783           // RFPC 0037
784           if (Value = 0.0) or (t_Expense.FieldByName('Transmission').AsFloat=0.0) then
785             FieldByName('Transmission').AsFloat := 0.0
786           else FieldByName('Transmission').AsFloat := (Value *
787   t_Expense.FieldByName('Transmission').AsFloat);
788           if (Value = 0.0) or (t_Expense.FieldByName('Switch').AsFloat=0.0) then
789             FieldByName('Switch').AsFloat := 0.0
790           else FieldByName('Switch').AsFloat := (Value * t_Expense.FieldByName('Switch').AsFloat);
791           if (Value = 0.0) or (t_Expense.FieldByName('Access').AsFloat=0.0) then
792             FieldByName('Access').AsFloat := 0.0
793           else FieldByName('Access').AsFloat := (Value * t_Expense.FieldByName('Access').AsFloat);
794           if (Value = 0.0) or (t_Expense.FieldByName('IOT').AsFloat=0.0) then
795             FieldByName('IOT').AsFloat := 0.0
796           else FieldByName('IOT').AsFloat := (Value * t_Expense.FieldByName('IOT').AsFloat);
797           if (Value = 0.0) or (t_Expense.FieldByName('Direct Other').AsFloat=0.0) then
798             FieldByName('Direct Other').AsFloat := 0.0
799           else FieldByName('Direct Other').AsFloat := (Value * t_Expense.FieldByName('Direct
800   Other').AsFloat);
801           if (Value = 0.0) or (t_Expense.FieldByName('Bill_Pay').AsFloat=0.0) then
802             FieldByName('B and C').AsFloat := 0.0
803           else FieldByName('B and C').AsFloat := (Value * t_Expense.FieldByName('Bill_Pay').AsFloat);
804           if (Value = 0.0) or (t_Expense.FieldByName('Oper_Serv').AsFloat=0.0) then
805             FieldByName('Oper').AsFloat := 0.0
806           else FieldByName('Oper').AsFloat := (Value * t_Expense.FieldByName('Oper_Serv').AsFloat);
807           if (Value = 0.0) or (t_Expense.FieldByName('PubCom').AsFloat=0.0) then
808             FieldByName('PUB COMM').AsFloat := 0.0
809           else FieldByName('PUB COMM').AsFloat := (Value * t_Expense.FieldByName('PubCom').AsFloat);
810           if (Value = 0.0) or (t_Expense.FieldByName('Consumer').AsFloat=0.0) then
811             FieldByName('Consumer').AsFloat := 0.0
812           else FieldByName('Consumer').AsFloat := (Value * t_Expense.FieldByName('Consumer').AsFloat);
813           if (Value = 0.0) or (t_Expense.FieldByName('Business').AsFloat=0.0) then
814             FieldByName('Business').AsFloat := 0.0
815           else FieldByName('Business').AsFloat := (Value * t_Expense.FieldByName('Business').AsFloat);
816           if (Value = 0.0) or (t_Expense.FieldByName('CXR').AsFloat=0.0) then
```

```
817               FieldByName('CXR').AsFloat := 0.0
818             else FieldByName('CXR').AsFloat := (Value * t_Expense.FieldByName('CXR').AsFloat);
819             if (Value = 0.0) or (t_Expense.FieldByName('Common').AsFloat=0.0) then
820               FieldByName('Common').AsFloat := 0.0
821             else FieldByName('Common').AsFloat := (Value * t_Expense.FieldByName('Common').AsFloat);
822           end;
823         Post;  // post the updated information
824       end; {with t_JointSharedCost}
825   end; {UpdateJointSharedCosts}
826
827   procedure Tf_Expense.CalcFactors;
828    Var
829      Value    : double;
830      ROR      : double;
831      Actno    : String;
832      SubTotal : Double;
833
834    begin
835     //RFPC0029 Run Calibration
836     Calibrate(Calibration);
837
838     { open Expense Investment database file (3002 - xxgteexp.db) }
839     TabName :=  f_associate.GetFileName(3002);
840     With t_Expense do
841       begin
842          if Active then Close;
843          DatabaseName := ExtractFilePath(TabName);
844          TableName := ExtractFilename(TabName);
845          TableType := ttDefault;
846          Exclusive := True;
847          Open;
848       end;
849
850     { open Plant Life database file (3005 - xxplntlf.db) }
851     TabName :=  f_associate.GetFileName(3005);
852     With t_ExpLife do
853       begin
854          if Active then Close;
855          DatabaseName := ExtractFilePath(TabName);
```

```
856         TableName := ExtractFilename(TabName);
857         TableType := ttDefault;
858         Exclusive := True;
859         Open;
860       end;
861
862     { open Expense Base database file (3006 - xxexbase.db) }
863     TabName :=  f_associate.GetFileName(3006);
864     With t_ExpBase do
865       begin
866         if Active then Close;
867         DatabaseName := ExtractFilePath(TabName);
868         TableName := ExtractFilename(TabName);
869         TableType := ttDefault;
870         Exclusive := True;
871         Open;
872       end;
873
874     TabName :=  f_associate.GetFileName(3010);
875     With t_PropTax do
876       begin
877         if Active then Close;
878         DatabaseName := ExtractFilePath(TabName);
879         TableName := ExtractFilename(TabName);
880         TableType := ttDefault;
881         Exclusive := True;
882         Open;
883       end;
884
885     t_RoExpense.Open;      // open Expense Run Time Options Table
886
887     WeAreWatching.LogEvent(1,'CalcFactors','');
888     WeAreWatching.LogEvent(2,'CalcFactors','Processing Expense Factors.');
889     { create an expense result table (EXPRSLT), set the fields}
890     with t_ExpResult do
891       begin
892         Active := False;
893         DatabaseName := 'modata';
894         TableName := 'EXPRSLT';
```

```
895         TableType := ttParadox;
896         Exclusive := True;
897         with FieldDefs do
898           begin
899             Clear;
900             Add ('Account',       FTString, 6, True);
901             Add ('Depreciation and Return',   FTFloat, 0, False);
902             Add ('Composite Income Tax', FTFloat, 0, False);
903             Add ('Maintenance and Support',   FTFloat, 0, False);
904             Add ('Property Tax',  FTFloat, 0, False);
905           end;
906         with IndexDefs do
907           begin
908             Clear;
909             Add('Main', 'Account', [ixPrimary, ixUnique]);
910           end;
911         CreateTable;
912         Open;
913       end;
914
915     // create the Joint/Shared Cost table (RFPC 0006) yv
916     Make_JointShared;  // table is open by default after creating fields and indexes
917     Make_Worksheets;   // table is open by default after creating fields and indexes
918
919     // get Run Time Expense 'Common Cost' Value
920     Common_Cost := t_RoExpense.FieldByName('Common Cost').AsFloat;
921
922     {From Run Time Expense Table determine if expense is 'retail' or 'wholesale'
923     and set Market flag accordingly }
924     if t_RoExpense.FieldByName('Market').AsString = 'Retail'
925       then Market := 'R'    // set Market flag to retail (R)
926       else Market := 'W';   // set Market flag to wholesale (W)
927     {RFPC 0006  determine if the shared flag has been enabled}
928     ExpShared:=t_RoExpense.FieldByName('Shared').AsBoolean;
929
930      {Calculate data for above table }
931      t_ExpBase.First;  // set to first record in Expense Base Table
932      t_ExpLife.First;  // set to first record in Plant Life Table
933
```

```
934     {Calculate the Rate On Return (ROR) from the Expense Base Table }
935     ROR := t_ExpBase.FieldByName('Long Term Fact').AsFloat *    // portion long-term
936            t_ExpBase.FieldByName('Long Term Rate').AsFloat +    // interest rate on long-term
937            t_ExpBase.FieldByName('Short Term Fact').AsFloat *   // portion short-term
938            t_ExpBase.FieldByName('Short Term Rate').AsFloat +   // interest rate on short-term
939            t_ExpBase.FieldByName('Common Fact').AsFloat *       // portion of common stock
940            t_ExpBase.FieldByName('Common Rate').AsFloat +       // cost of financing w/ common stock
941            t_ExpBase.FieldByName('PRFD Fact').AsFloat *         // portion of preferred stock
942            t_ExpBase.FieldByName('PRFD Rate').AsFloat;          // expected cost of preferred stock
943
944   WeAreWatching.LogEvent(4,'CalcFactors',' Market: '+Market+'   Common Cost: '+
945                       FloatToStr(Common_Cost)+'   ROR: '+FloatToStr(ROR));
946
947     while not t_ExpLife.EOF do
948       begin
949         t_ExpResult.Insert;  // create a new, empty record in expense result table
950         // set Account name in result table
951         t_ExpResult.FieldByName('Account').AsString := t_ExpLife.FieldByName('Account').AsString;
952         AccountStr:=t_ExpLife.FieldByName('Account').AsString;
953         WeAreWatching.LogEvent(4,'CalcFactors',' Account: '+t_ExpResult.FieldByName('Account').AsString+
954                         '  Market: '+t_RoExpense.FieldByName('Life').AsString);
955         {Calculate the Depreciation and Return and Composite Income Tax using the Book values
956         (life of asset as prescribed by regulatory commissions) or the Ecomonic
957         values (actual viable life of plant). Macrs Years = life of plant
958         according to the Modified Accelerated Depreciation and Return System provided
959         by the IRS.}
960         if t_RoExpense.FieldByName('Life').AsString = 'Book'
961           then begin
962                  t_ExpResult.FieldByName('Depreciation and Return').AsFloat :=
963                     DepreciationAndReturn ( ROR, t_ExpLife.FieldByName('Book Life').AsFloat,
964                       t_ExpLife.FieldByName('Book Salvage').AsFloat);  //RFPC0051
965                  t_ExpResult.FieldByName('Composite Income Tax').AsFloat :=
966                     CompositeIncomeTax ( ROR, t_ExpLife.FieldByName('Book Life').AsFloat,
967                                          t_ExpLife.FieldByName('Book Salvage').AsFloat,
968                                          t_ExpLife.FieldByName('Macrs Years').AsInteger); //RFPC0051
969                end
970           else begin
971                  t_ExpResult.FieldByName('Depreciation and Return').AsFloat :=
972                     DepreciationAndReturn ( ROR, t_ExpLife.FieldByName('Economic Life').AsFloat,
```

```
973                     t_ExpLife.FieldByName('Econ Salvage').AsFloat); //RFPC0051
974
975                 t_ExpResult.FieldByName('Composite Income Tax').AsFloat :=
976                    CompositeIncomeTax ( ROR, t_ExpLife.FieldByName('Economic Life').AsFloat,
977                                         t_ExpLife.FieldByName('Econ Salvage').AsFloat,
978                                         t_ExpLife.FieldByName('Macrs Years').AsInteger); //RFPC0051
979               end;
980        t_ExpResult.Post;  // write changes to expense result table
981        WeAreWatching.LogEvent(4,'CalcFactors','  Depreciation and Return = '+
982                                        t_ExpResult.FieldByName('Depreciation and Return').AsString);
983        WeAreWatching.LogEvent(4,'CalcFactors','  Composite Income Tax = '+
984                                        FloatToStr(t_ExpResult.FieldByName('Composite Income Tax').AsFloat));
985
986        t_ExpLife.Next;    // get next record in Plant Life table
987      end;
988
989    { Initialize expense items }
990    Exp_Cable        := 0.0;
991    Exp_FiAer        := 0.0;
992    Exp_FiBur        := 0.0;
993    Exp_FiUnd        := 0.0;
994    Exp_CuAer        := 0.0;
995    Exp_CuBur        := 0.0;
996    Exp_CuUnd        := 0.0;
997    Exp_Poles        := 0.0;
998    Exp_Conduit      := 0.0;
999    Exp_Transmission := 0.0;
1000   Exp_Switch       := 0.0;
1001   Exp_Access       := 0.0;
1002   Exp_IOT          := 0.0;
1003   Exp_OD           := 0.0;
1004   Exp_BillPay      := 0.0;
1005   Exp_OperSvc      := 0.0;
1006   Exp_PubCom       := 0.0;
1007   Exp_Consumer     := 0.0;
1008   Exp_Business     := 0.0;
1009   Exp_IXC          := 0.0;
1010   Exp_Common       := 0.0;
1011
```

```
1012      // identify field(s) within table in which to search
1013      t_Expense.IndexFieldNames := 'Category;Account';
1014      t_Expense.CancelRange;  // clear all current search ranges
1015      { set search range for expense market value (R or W) and accounts 600000 thru 724999 }
1016      t_Expense.SetRange([ Market,'600000'], [Market,'724999']);
1017      t_Expense.first;  // go to first record in table
1018      While not t_Expense.EOF do
1019        begin
1020          {calculate the expense value, relating to the current table account
1021          information, using 'Amount' (data per FCC ARMIS reporting), 'Adjust1'
1022          (for retail (avoided cost), billing & collection, directory cost,
1023          and non-recurring activities), 'Adjust2' (for expenses identified
1024          in the activity-based costing approach), & 'Adjust3' (to eliminate
1025          an account balance, adjust plant balances, or reflect inflation
1026          and/or productivity gains. }
1027          Value       := t_Expense.FieldByName('Amount').AsFloat *
1028           (1.0 - (1.0 - t_Expense.FieldByName('Adjust1').AsFloat) -
1029                  (1.0 - t_Expense.FieldByName('Adjust2').AsFloat)) *
1030                         t_Expense.FieldByName('Adjust3').AsFloat;
1031          {relate the expense value to Inflation, Productivity, & Horizon }
1032          Value        := Value * exp(Ln((1.0+t_RoExpense.FieldByName('Inflation').AsFloat) /
1033                                         (1.0+t_RoExpense.FieldByName('Productivity').AsFloat)) *
1034                                          t_RoExpense.FieldByName('Horizon').AsFloat);
1035
1036          {update the Joint/Shared Cost table with the calculated value RFPC 0006}
1037          ActNo:=Copy(t_Expense.FieldByName('Account').AsString,1,4);
1038          UpdateJointSharedCosts(Market,ActNo,Value);
1039          if (ActNo >= '6112') and (ActNo <= '6124') then
1040            UpdateJointSharedCosts(Market,'Z'+ActNo,Value);  // to be used in network services shared cost
1041  report
1042          {calculate new item expenses using adjusted expense value }
1043          Exp_Cable        := Exp_Cable + Value * t_Expense.FieldByName('Cable').AsFloat;
1044          Exp_FiAer        := Exp_FiAer + Value * t_Expense.FieldByName('FiAer').AsFloat;
1045          Exp_FiBur        := Exp_FiBur + Value * t_Expense.FieldByName('FiBur').AsFloat;
1046          Exp_FiUnd        := Exp_FiUnd + Value * t_Expense.FieldByName('FiUnd').AsFloat;
1047          Exp_CuAer        := Exp_CuAer + Value * t_Expense.FieldByName('CuAer').AsFloat;
1048          Exp_CuBur        := Exp_CuBur + Value * t_Expense.FieldByName('CuBur').AsFloat;
1049          Exp_CuUnd        := Exp_CuUnd + Value * t_Expense.FieldByName('CuUnd').AsFloat;
1050
```

```
1051        Exp_Poles        := Exp_Poles + Value * t_Expense.FieldByName('Poles').AsFloat;
1052        Exp_Conduit      := Exp_Conduit + Value * t_Expense.FieldByName('Conduit').AsFloat;
1053        // RFPC 0037
1054        Exp_Transmission := Exp_Transmission + Value * t_Expense.FieldByName('Transmission').AsFloat;
1055        Exp_Switch       := Exp_Switch + Value * t_Expense.FieldByName('Switch').AsFloat;
1056        Exp_Access       := Exp_Access + Value * t_Expense.FieldByName('Access').AsFloat;
1057        Exp_IOT          := Exp_IOT + Value * t_Expense.FieldByName('IOT').AsFloat;
1058        Exp_OD           := Exp_OD + Value * t_Expense.FieldByName('Direct Other').AsFloat;
1059        Exp_BillPay      := Exp_BillPay + Value * t_Expense.FieldByName('Bill_Pay').AsFloat;
1060        Exp_OperSvc      := Exp_OperSvc + Value * t_Expense.FieldByName('Oper_Serv').AsFloat;
1061        Exp_PubCom       := Exp_PubCom + Value * t_Expense.FieldByName('PubCom').AsFloat;
1062        Exp_Consumer     := Exp_Consumer + Value * t_Expense.FieldByName('Consumer').AsFloat;
1063        Exp_Business     := Exp_Business + Value * t_Expense.FieldByName('Business').AsFloat;
1064        Exp_IXC          := Exp_IXC + Value * t_Expense.FieldByName('CXR').AsFloat;
1065        Exp_Common       := Exp_Common + Value * t_Expense.FieldByName('Common').AsFloat;
1066        t_Expense.next;
1067    end;
1068
1069    t_Expense.CancelRange;  // clear all current search ranges
1070    t_Expense.SetRange([Market,'200000'], [Market,'299999']); // set new search range
1071    t_Expense.first;        // set to beginning of table
1072
1073    // set the range for GSA
1074    t_PropTax.CancelRange;
1075    t_PropTax.SetRange(['211100'], ['212499']);
1076    t_PropTax.first;
1077
1078    {calculate more item expenses within account range of 200000 to 299999.
1079    using amount and adjustment values in record. }
1080    While not t_Expense.EOF do
1081      begin
1082        Value := t_Expense.FieldByName('Amount').AsFloat * t_Expense.FieldByName('Adjust1').AsFloat*
1083                 t_Expense.FieldByName('Adjust2').AsFloat * t_Expense.FieldByName('Adjust3').AsFloat;
1084        if t_ExpResult.FindKey([t_Expense.FieldByName('Account').AsString])
1085          then begin
1086                 if t_PropTax.FindKey([t_Expense.FieldByName('Account').AsString])
1087                   then begin
1088                          Value := Value * (t_ExpResult.FieldByName('Depreciation and Return').AsFloat +
1089                                            t_ExpResult.FieldByName('Composite Income Tax').AsFloat+
```

```
1090                                                (t_PropTax.FieldByName('Property Tax Flag').AsInteger*
1091                                                 t_PropTax.FieldByName('Property Tax').AsFloat));
1092                              { update the Joint costs for 61xx accounts}
1093                              ActNo:='6'+Copy(t_Expense.FieldByName('Account').AsString,2,3);
1094                              if (Copy(t_Expense.FieldByName('Account').AsString,1,4) = '2111') then
1095                                ActNo:='6121';
1096                              UpdateJointSharedCosts(Market,ActNo,Value);
1097                              ActNo:=Copy(t_Expense.FieldByName('Account').AsString,1,4);
1098                              UpdateJointSharedCosts(Market,ActNo,Value);
1099                            end;
1100                  end
1101              else begin
1102                    messagedlg('EXPENSE - Missing account code '+ t_Expense.FieldByName('Account').AsString,
1103                               mtError, [mbOk], 0 );
1104                    WeAreWatching.LogEvent(1,'CalcFactors','* Missing Account Code: '+
1105  t_Expense.FieldByName('Account').AsString);
1106                  end;
1107
1108           { calculate new item expenses using the adjusted expense value }
1109           Exp_Cable        := Exp_Cable + Value * t_Expense.FieldByName('Cable').AsFloat;
1110           Exp_FiAer        := Exp_FiAer + Value * t_Expense.FieldByName('FiAer').AsFloat;
1111           Exp_FiBur        := Exp_FiBur + Value * t_Expense.FieldByName('FiBur').AsFloat;
1112           Exp_FiUnd        := Exp_FiUnd + Value * t_Expense.FieldByName('FiUnd').AsFloat;
1113           Exp_CuAer        := Exp_CuAer + Value * t_Expense.FieldByName('CuAer').AsFloat;
1114           Exp_CuBur        := Exp_CuBur + Value * t_Expense.FieldByName('CuBur').AsFloat;
1115           Exp_CuUnd        := Exp_CuUnd + Value * t_Expense.FieldByName('CuUnd').AsFloat;
1116
1117           Exp_Poles        := Exp_Poles + Value * t_Expense.FieldByName('Poles').AsFloat;
1118           Exp_Conduit      := Exp_Conduit + Value * t_Expense.FieldByName('Conduit').AsFloat;
1119           // RFPC 0037
1120           Exp_Transmission := Exp_Transmission + Value * t_Expense.FieldByName('Transmission').AsFloat;
1121           Exp_Switch       := Exp_Switch + Value * t_Expense.FieldByName('Switch').AsFloat;
1122           Exp_Access       := Exp_Access + Value * t_Expense.FieldByName('Access').AsFloat;
1123           Exp_IOT          := Exp_IOT + Value * t_Expense.FieldByName('IOT').AsFloat;
1124           Exp_OD           := Exp_OD + Value * t_Expense.FieldByName('Direct Other').AsFloat;
1125           Exp_BillPay      := Exp_BillPay + Value * t_Expense.FieldByName('Bill_Pay').AsFloat;
1126           Exp_OperSvc      := Exp_OperSvc + Value * t_Expense.FieldByName('Oper_Serv').AsFloat;
1127           Exp_PubCom       := Exp_PubCom + Value * t_Expense.FieldByName('PubCom').AsFloat;
1128           Exp_Consumer     := Exp_Consumer + Value * t_Expense.FieldByName('Consumer').AsFloat;
```

```
1129          Exp_Business     := Exp_Business + Value * t_Expense.FieldByName('Business').AsFloat;
1130          Exp_IXC          := Exp_IXC + Value * t_Expense.FieldByName('CXR').AsFloat;
1131          Exp_Common       := Exp_Common + Value * t_Expense.FieldByName('Common').AsFloat;
1132          t_Expense.next;
1133        end;
1134
1135      Value :=  Exp_Cable + Exp_FiAer + Exp_FiBur + Exp_FiUnd +
1136                Exp_CuAer + Exp_CuBur + Exp_CuUnd +
1137                Exp_Poles+Exp_Conduit+Exp_Transmission+
1138                Exp_Switch+Exp_IOT;
1139      if (Exp_OD > 0.0) and (Value > 0.0)
1140        then begin
1141               Exp_FiAer        :=  Exp_FiAer    + Exp_OD * Exp_FiAer     / Value;
1142               Exp_FiBur        :=  Exp_FiBur    + Exp_OD * Exp_FiBur     / Value;
1143               Exp_FiUnd        :=  Exp_FiUnd    + Exp_OD * Exp_FiUnd     / Value;
1144               Exp_CuAer        :=  Exp_CuAer    + Exp_OD * Exp_CuAer     / Value;
1145               Exp_CuBur        :=  Exp_CuBur    + Exp_OD * Exp_CuBur     / Value;
1146               Exp_CuUnd        :=  Exp_CuUnd    + Exp_OD * Exp_CuUnd     / Value;
1147
1148               Exp_Cable        := Exp_Cable     + Exp_OD * Exp_Cable     / Value;
1149               Exp_Poles        := Exp_Poles     + Exp_OD * Exp_Poles     / Value;
1150               Exp_Conduit      := Exp_Conduit   + Exp_OD * Exp_Conduit   / Value;
1151               Exp_Transmission := Exp_Transmission + Exp_OD * Exp_Transmission / Value;
1152               Exp_Switch       := Exp_Switch    + Exp_OD * Exp_Switch    / Value;
1153               Exp_IOT          := Exp_IOT       + Exp_OD * Exp_IOT       / Value;
1154             end;
1155
1156      {added 7/18/97 per Roger (DDEAN)----------------------}
1157      { get the percentage of the total expenses in its category}
1158      Value:= Exp_FiAer +Exp_FiBur +Exp_FiUnd +Exp_CuAer +Exp_CuBur +Exp_CuUnd;
1159      if (Exp_Cable > 0.0) and (Value > 0.0)
1160       then begin
1161              Exp_FiAer    :=  Exp_FiAer + Exp_Cable * Exp_FiAer  / Value;
1162              Exp_FiBur    :=  Exp_FiBur + Exp_Cable * Exp_FiBur  / Value;
1163              Exp_FiUnd    :=  Exp_FiUnd + Exp_Cable * Exp_FiUnd  / Value;
1164              Exp_CuAer    :=  Exp_CuAer + Exp_Cable * Exp_CuAer  / Value;
1165              Exp_CuBur    :=  Exp_CuBur + Exp_Cable * Exp_CuBur  / Value;
1166              Exp_CuUnd    :=  Exp_CuUnd + Exp_Cable * Exp_CuUnd  / Value;
1167            end;
```

```
1168      {end add-------------------------------------------------------}
1169
1170      // RFPC 0006
1171      // Redistributed totals assigned to the redistribution array
1172      FillChar(RedistributeTot, SizeOf(ArrayDouble),0);
1173      RedistributeTot[4] :=Exp_FiAer;
1174      RedistributeTot[5] :=Exp_CuAer;
1175      RedistributeTot[6] :=Exp_FiBur;
1176      RedistributeTot[7] :=Exp_CuBur;
1177      RedistributeTot[8] :=Exp_FiUnd;
1178      RedistributeTot[9] :=Exp_CuUnd;
1179      RedistributeTot[10]:=Exp_Poles;
1180      RedistributeTot[11]:=Exp_Conduit;
1181      RedistributeTot[12]:=Exp_Transmission;
1182      RedistributeTot[13]:=Exp_Switch;
1183      RedistributeTot[14]:=Exp_IOT;
1184
1185      {added 7/18/97 per Roger (DDEAN) ----------------------------}
1186      {get the total expense for each account category}
1187
1188      t_Expense.CancelRange;
1189      t_Expense.first;
1190      { Aerial Copper: 242110 + 242610;    RFPC 0025 (YV)}
1191      Inv_CuAer := 0;
1192      if t_Expense.FindKey([Market,'242110']) then
1193        Inv_CuAer := Inv_CuAer+(t_Expense.FieldByName('Amount').AsFloat  *
1194                                t_Expense.FieldByName('Adjust1').AsFloat *
1195                                t_Expense.FieldByName('Adjust2').AsFloat *
1196                                t_Expense.FieldByName('Adjust3').AsFloat);
1197      if t_Expense.FindKey([Market,'242610']) then
1198        Inv_CuAer := Inv_CuAer+(t_Expense.FieldByName('Amount').AsFloat  *
1199                                t_Expense.FieldByName('Adjust1').AsFloat *
1200                                t_Expense.FieldByName('Adjust2').AsFloat *
1201                                t_Expense.FieldByName('Adjust3').AsFloat);
1202      { Buried Copper: 242310 + 242410 + 242510;   RFPC 0025 (YV)}
1203      Inv_CuBur := 0;
1204      if t_Expense.FindKey([Market,'242310']) then
1205        Inv_CuBur :=  Inv_CuBur+(t_Expense.FieldByName('Amount').AsFloat  *
1206                                 t_Expense.FieldByName('Adjust1').AsFloat *
```

```
1207                              t_Expense.FieldByName('Adjust2').AsFloat *
1208                              t_Expense.FieldByName('Adjust3').AsFloat);
1209      if t_Expense.FindKey([Market,'242410']) then
1210        Inv_CuBur :=  Inv_CuBur+(t_Expense.FieldByName('Amount').AsFloat  *
1211                              t_Expense.FieldByName('Adjust1').AsFloat *
1212                              t_Expense.FieldByName('Adjust2').AsFloat *
1213                              t_Expense.FieldByName('Adjust3').AsFloat);
1214      if t_Expense.FindKey([Market,'242510']) then
1215        Inv_CuBur :=  Inv_CuBur+(t_Expense.FieldByName('Amount').AsFloat  *
1216                              t_Expense.FieldByName('Adjust1').AsFloat *
1217                              t_Expense.FieldByName('Adjust2').AsFloat *
1218                              t_Expense.FieldByName('Adjust3').AsFloat);
1219      {Underground Copper: 242210}
1220      if t_Expense.FindKey([Market,'242210']) then
1221        Inv_CuUnd :=  t_Expense.FieldByName('Amount').AsFloat  *
1222                      t_Expense.FieldByName('Adjust1').AsFloat *
1223                      t_Expense.FieldByName('Adjust2').AsFloat *
1224                      t_Expense.FieldByName('Adjust3').AsFloat;
1225
1226      { Aerial Fiber: 242120 + 242620;   RFPC 0025 (YV)}
1227      Inv_FiAer := 0;
1228      if t_Expense.FindKey([Market,'242120']) then
1229        Inv_FiAer :=  Inv_FiAer+(t_Expense.FieldByName('Amount').AsFloat  *
1230                              t_Expense.FieldByName('Adjust1').AsFloat *
1231                              t_Expense.FieldByName('Adjust2').AsFloat *
1232                              t_Expense.FieldByName('Adjust3').AsFloat);
1233      if t_Expense.FindKey([Market,'242620']) then
1234        Inv_FiAer :=  Inv_FiAer+(t_Expense.FieldByName('Amount').AsFloat  *
1235                              t_Expense.FieldByName('Adjust1').AsFloat *
1236                              t_Expense.FieldByName('Adjust2').AsFloat *
1237                              t_Expense.FieldByName('Adjust3').AsFloat);
1238      { buried Fiber: 242320 + 242420 + 242520;   RFPC 0025 (YV)}
1239      Inv_FiBur := 0;
1240      if t_Expense.FindKey([Market,'242320']) then
1241        Inv_FiBur :=  Inv_FiBur+(t_Expense.FieldByName('Amount').AsFloat  *
1242                              t_Expense.FieldByName('Adjust1').AsFloat *
1243                              t_Expense.FieldByName('Adjust2').AsFloat *
1244                              t_Expense.FieldByName('Adjust3').AsFloat);
1245      if t_Expense.FindKey([Market,'242420']) then
```

```
1246        Inv_FiBur :=  Inv_FiBur+(t_Expense.FieldByName('Amount').AsFloat  *
1247                                 t_Expense.FieldByName('Adjust1').AsFloat *
1248                                 t_Expense.FieldByName('Adjust2').AsFloat *
1249                                 t_Expense.FieldByName('Adjust3').AsFloat);
1250      if t_Expense.FindKey([Market,'242520']) then
1251        Inv_FiBur :=  Inv_FiBur+(t_Expense.FieldByName('Amount').AsFloat  *
1252                                 t_Expense.FieldByName('Adjust1').AsFloat *
1253                                 t_Expense.FieldByName('Adjust2').AsFloat *
1254                                 t_Expense.FieldByName('Adjust3').AsFloat);
1255      { Underground Fiber: 242220}
1256      Inv_FiUnd:=0;
1257      if t_Expense.FindKey([Market,'242220']) then
1258        Inv_FiUnd :=  Inv_FiUnd+(t_Expense.FieldByName('Amount').AsFloat  *
1259                                 t_Expense.FieldByName('Adjust1').AsFloat *
1260                                 t_Expense.FieldByName('Adjust2').AsFloat *
1261                                 t_Expense.FieldByName('Adjust3').AsFloat);
1262      {end add-------------------------------------------------------}
1263
1264      { Poles: 241099 - 241199 }
1265      t_Expense.CancelRange;
1266      t_Expense.SetRange([Market,'241099'], [Market,'241199']);
1267      t_Expense.first;
1268
1269      Inv_Poles := 0.0;  // initialize to 0 for multiple account summing
1270      While not t_Expense.EOF do
1271        begin
1272          Inv_Poles    := Inv_Poles + (t_Expense.FieldByName('Amount').AsFloat *
1273                      t_Expense.FieldByName('Adjust1').AsFloat) *
1274                      t_Expense.FieldByName('Adjust2').AsFloat *
1275                      t_Expense.FieldByName('Adjust3').AsFloat;
1276          t_Expense.next;
1277        end;
1278
1279      { Conduit: 244099 - 244199 }
1280      t_Expense.CancelRange;
1281      t_Expense.SetRange([Market,'244099'], [Market,'244199']);
1282      t_Expense.first;
1283
1284      Inv_Conduit := 0.0;  // initialize to 0 for multiple account summing
```

```
1285    While not t_Expense.EOF do
1286      begin
1287        Inv_Conduit  := Inv_Conduit + (t_Expense.FieldByName('Amount').AsFloat *
1288                    t_Expense.FieldByName('Adjustl').AsFloat) *
1289                    t_Expense.FieldByName('Adjust2').AsFloat *
1290                    t_Expense.FieldByName('Adjust3').AsFloat;
1291        t_Expense.next;
1292      end;
1293
1294    { Transmission: 223099 - 223299 }
1295    t_Expense.CancelRange;
1296    t_Expense.SetRange([Market,'223099'], [Market,'223299']);
1297    t_Expense.first;
1298
1299    Inv_Transmission := 0.0;  // initialize to 0 for multiple account summing
1300    While not t_Expense.EOF do
1301      begin
1302       Inv_Transmission:= Inv_transmission + (t_Expense.FieldByName('Amount').AsFloat *
1303                          t_Expense.FieldByName('Adjust1').AsFloat) *
1304                          t_Expense.FieldByName('Adjust2').AsFloat *
1305                          t_Expense.FieldByName('Adjust3').AsFloat;
1306        t_Expense.next;
1307      end;
1308
1309    { Switch: 221200 - 221299 }
1310    t_Expense.CancelRange;
1311    t_Expense.SetRange([Market,'221200'], [Market,'221299']);
1312    t_Expense.first;
1313
1314    Inv_Switch := 0.0;  // initialize to 0 for multiple account summing
1315    While not t_Expense.EOF do
1316      begin
1317        Inv_Switch   := Inv_Switch + (t_Expense.FieldByName('Amount').AsFloat *
1318                    t_Expense.FieldByName('Adjust1').AsFloat) *
1319                    t_Expense.FieldByName('Adjust2').AsFloat *
1320                    t_Expense.FieldByName('Adjust3').AsFloat;
1321        t_Expense.next;
1322      end;
1323
```

```
1324      //RFPC0065 add account 2690 to switch
1325      t_Expense.CancelRange;
1326      t_Expense.SetRange([Market,'269000'], [Market,'269099']);
1327      t_Expense.first;
1328
1329      While not t_Expense.EOF do
1330        begin
1331          Inv_Switch   := Inv_Switch + (t_Expense.FieldByName('Amount').AsFloat *
1332                      t_Expense.FieldByName('Adjustl').AsFloat) *
1333                      t_Expense.FieldByName('Adjust2').AsFloat *
1334                      t_Expense.FieldByName('Adjust3').AsFloat;
1335          t_Expense.next;
1336        end;
1337
1338      // RFPC 0006
1339      // save out the investment totals
1340      with t_JointSharedCost do
1341        begin
1342          Append;  // add a new record
1343          FieldByName('Account').AsString     := 'C) Total Investment';
1344          FieldByName('Category').AsString    := Market;
1345          FieldByName('Cable').AsFloat := Inv_Cable;
1346          FieldByName('Aerial Non-Metallic').AsFloat := Inv_FiAer;
1347          FieldByName('Buried Non-Metallic').AsFloat := Inv_FiBur;
1348          FieldByName('Underground Non-Metallic').AsFloat := Inv_FiUnd;
1349          FieldByName('Aerial Metallic').AsFloat := Inv_CuAer;
1350          FieldByName('Buried Metallic').AsFloat := Inv_CuBur;
1351          FieldByName('Underground Metallic').AsFloat := Inv_CuUnd;
1352          FieldByName('Poles').AsFloat := Inv_Poles;
1353          FieldByName('Conduit').AsFloat := Inv_Conduit;
1354          FieldByName('Transmission').AsFloat := Inv_Transmission;
1355          FieldByName('Switch').AsFloat := Inv_Switch;
1356          Post;   // post the new record
1357        end; {with t_JointSharedCost}
1358      // aggregate the joint shared subtotals, direct expense totals, compute the
1359      // ratios and save everything out to the joint shared table.  Close the table
1360      // when done
1361      DirectExpAndRatios(Market);
1362
```

```
1363      t_ExpResult.IndexFieldNames := 'Account';  {set index on account #}
1364
1365      {added 10/7/98 per Joe Abs (YVashee)----------------------}
1366      {GSA}
1367      t_ExpResult.CancelRange;
1368      t_ExpResult.SetRange(['211100'],['212499']);   {GSA}
1369      t_ExpResult.First;
1370      while not t_ExpResult.EOF do begin
1371        t_ExpResult.Edit;
1372        if t_PropTax.FindKey([t_ExpResult.FieldByName('Account')]) and
1373           (t_PropTax.FieldByName('Property Tax Flag').AsInteger = 1) then
1374         t_ExpResult.FieldByName('Property Tax').AsFloat :=
1375                         t_PropTax.FieldByName('Property Tax').AsFloat
1376        else t_ExpResult.FieldByName('Property Tax').AsString := '';
1377        t_ExpResult.Post;
1378        t_ExpResult.Next;
1379      end;
1380
1381      {added 7/18/97 per Roger (DDEAN)----------------------}
1382      {GENERATE EXPENSE RESULT FILE}
1383      {AERIAL COPPER}
1384      t_ExpResult.CancelRange;
1385      t_ExpResult.SetRange(['242110'],['242110']);   {aerial copper}
1386      t_ExpResult.First;    // set to first record in table
1387      if not ExpShared then Value:=Ratios[5]  // RFPC 0006
1388      else Value := Exp_CuAer/Inv_CuAer;  // % of aerial copper expense
1389      while not t_ExpResult.EOF do begin
1390        t_ExpResult.Edit;   // allow record to be changed (edited)
1391        t_ExpResult.FieldByName('Maintenance and Support').AsFloat := Value;
1392        t_ExpResult.FieldByName('Property Tax').AsFloat :=
1393                         t_ExpBase.FieldByName('Property Tax').AsFloat;
1394        t_ExpResult.Post;   // write new data to table
1395        t_ExpResult.Next;   // next record in table
1396      end;  // loop until end of file is encountered
1397
1398      {AERIAL FIBER}
1399      t_ExpResult.CancelRange;
1400      t_ExpResult.SetRange(['242120'], ['242120']);
1401      t_ExpResult.First;
```

```
1402      if not ExpShared then Value:=Ratios[4]      // RFPC 0006
1403      else if Inv_FiAer <> 0.0 then Value := Exp_FiAer/Inv_FiAer
1404      else Value := 0.0;
1405      while not t_ExpResult.EOF do begin
1406        t_ExpResult.Edit;
1407        t_ExpResult.FieldByName('Maintenance and Support').AsFloat := Value;
1408        t_ExpResult.FieldByName('Property Tax').AsFloat :=
1409                        t_ExpBase.FieldByName('Property Tax').AsFloat;
1410        t_ExpResult.Post;
1411        t_ExpResult.Next;
1412      end;
1413
1414      {UNDERGROUND COPPER}
1415      t_ExpResult.CancelRange;
1416      t_ExpResult.SetRange(['242210'], ['242210']);
1417      t_ExpResult.First;
1418      if not ExpShared then Value:=Ratios[9]      // RFPC 0006
1419      else if Inv_CuUnd <> 0.0 then Value := Exp_CuUnd/Inv_CuUnd
1420      else Value := 0.0;
1421      while not t_ExpResult.EOF do begin
1422        Actno := t_ExpResult.FieldByName('Account').AsString;
1423        t_ExpResult.Edit;
1424        t_ExpResult.FieldByName('Maintenance and Support').AsFloat := Value;
1425        t_ExpResult.FieldByName('Property Tax').AsFloat :=
1426                        t_ExpBase.FieldByName('Property Tax').AsFloat;
1427        t_ExpResult.Post;
1428        t_ExpResult.Next;
1429      end;
1430
1431      {UNDERGROUND FIBER}
1432      t_ExpResult.CancelRange;
1433      t_ExpResult.SetRange(['242220'], ['242220']);
1434      t_ExpResult.First;
1435      if not ExpShared then Value:=Ratios[8]      // RFPC 0006
1436      else if Inv_FiUnd <> 0.0 then Value := Exp_FiUnd/Inv_FiUnd
1437      else Value := 0.0;
1438      while not t_ExpResult.EOF do begin
1439        Actno := t_ExpResult.FieldByName('Account').AsString;
1440        t_ExpResult.Edit;
```

```
1441        t_ExpResult.FieldByName('Maintenance and Support').AsFloat := Value;
1442        t_ExpResult.FieldByName('Property Tax').AsFloat :=
1443                        t_ExpBase.FieldByName('Property Tax').AsFloat;
1444        t_ExpResult.Post;
1445        t_ExpResult.Next;
1446      end;
1447
1448      {BURIED COPPER}
1449      t_ExpResult.CancelRange;
1450      t_ExpResult.SetRange(['242310'], ['242310']);
1451      t_ExpResult.First;
1452      if not ExpShared then Value:=Ratios[7]      // RFPC 0006
1453      else if Inv_CuBur <> 0.0 then Value := Exp_CuBur/Inv_CuBur
1454      else Value := 0.0;
1455      while not t_ExpResult.EOF do begin
1456        Actno := t_ExpResult.FieldByName('Account').AsString;
1457        t_ExpResult.Edit;
1458        t_ExpResult.FieldByName('Maintenance and Support').AsFloat := Value;
1459        t_ExpResult.FieldByName('Property Tax').AsFloat :=
1460                        t_ExpBase.FieldByName('Property Tax').AsFloat;
1461        t_ExpResult.Post;
1462        t_ExpResult.Next;
1463      end;
1464
1465      {BURIED FIBER}
1466      t_ExpResult.CancelRange;
1467      t_ExpResult.SetRange(['242320'], ['242320']);
1468      t_ExpResult.First;
1469      if not ExpShared then Value:=Ratios[6]      // RFPC 0006
1470      else if Inv_FiBur <> 0.0 then Value := Exp_FiBur/Inv_FiBur
1471      else Value := 0.0;
1472      while not t_ExpResult.EOF do begin
1473        Actno := t_ExpResult.FieldByName('Account').AsString;
1474        t_ExpResult.Edit;
1475        t_ExpResult.FieldByName('Maintenance and Support').AsFloat := Value;
1476        t_ExpResult.FieldByName('Property Tax').AsFloat :=
1477                        t_ExpBase.FieldByName('Property Tax').AsFloat;
1478        t_ExpResult.Post;
1479        t_ExpResult.Next;
```

```
1480      end;
1481      {End Add ------------------------------------------------------------------}
1482
1483      {POLES}
1484      t_ExpResult.CancelRange;
1485      t_ExpResult.SetRange(['241099'], ['241199']);
1486      t_ExpResult.first;
1487
1488      if not ExpShared then Value:=Ratios[10]      // RFPC 0006
1489      else if Inv_Poles <> 0.0 then Value := Exp_Poles/Inv_Poles
1490      else Value := 0.0;
1491      While not t_ExpResult.EOF do
1492        begin
1493          Actno := t_ExpResult.FieldByName('Account').AsString;
1494          t_ExpResult.Edit;
1495          t_ExpResult.FieldByName('Maintenance and Support').AsFloat := Value;
1496          t_ExpResult.FieldByName('Property Tax').AsFloat := t_ExpBase.FieldByName('Property Tax').AsFloat;
1497          t_ExpResult.Post;
1498          t_ExpResult.next;
1499        end;
1500
1501      {CONDUIT}
1502      t_ExpResult.CancelRange;
1503      t_ExpResult.SetRange(['244099'], ['244199']);
1504      t_ExpResult.first;
1505
1506      if not ExpShared then Value:=Ratios[11]      // RFPC 0006
1507      else if Inv_Conduit <> 0.0 then Value := Exp_Conduit/Inv_Conduit
1508      else Value := 0.0;
1509      While not t_ExpResult.EOF do
1510        begin
1511          Actno := t_ExpResult.FieldByName('Account').AsString;
1512          t_ExpResult.Edit;
1513          t_ExpResult.FieldByName('Maintenance and Support').AsFloat := Value;
1514          t_ExpResult.FieldByName('Property Tax').AsFloat := t_ExpBase.FieldByName('Property Tax').AsFloat;
1515          t_ExpResult.Post;
1516          t_ExpResult.next;
1517       end;
1518
```

```
1519      {TRANSMISSION}
1520      t_ExpResult.CancelRange;
1521      t_ExpResult.SetRange(['223099'], ['223299']);
1522      t_ExpResult.first;
1523
1524      if not ExpShared then Value:=Ratios[12]     // RFPC 0006
1525      else if Inv_Transmission <> 0.0 then Value := Exp_Transmission/Inv_Transmission
1526      else Value := 0.0;
1527      While not t_ExpResult.EOF do
1528        begin
1529          Actno := t_ExpResult.FieldByName('Account').AsString;
1530          t_ExpResult.Edit;
1531          t_ExpResult.FieldByName('Maintenance and Support').AsFloat := Value;
1532          t_ExpResult.FieldByName('Property Tax').AsFloat := t_ExpBase.FieldByName('Property Tax').AsFloat;
1533          t_ExpResult.Post;
1534          t_ExpResult.next;
1535        end;
1536
1537      {SWITCH}
1538      t_ExpResult.CancelRange;
1539      t_ExpResult.SetRange(['221200'], ['221299']);  {leave the analog switch}
1540      t_ExpResult.first;
1541
1542      if not ExpShared then Value:=Ratios[13]     // RFPC 0006
1543      else if Inv_Switch <> 0.0 then Value := Exp_Switch/Inv_Switch
1544      else Value := 0.0;
1545      While not t_ExpResult.EOF do
1546        begin
1547          Actno := t_ExpResult.FieldByName('Account').AsString;
1548          t_ExpResult.Edit;
1549          t_ExpResult.FieldByName('Maintenance and Support').AsFloat := Value;
1550          t_ExpResult.FieldByName('Property Tax').AsFloat := t_ExpBase.FieldByName('Property Tax').AsFloat;
1551          t_ExpResult.Post;
1552          t_ExpResult.next;
1553        end;
1554
1555      {RTU}                                       //RFPC0065 RTU
1556      t_PropTax.CancelRange;                      //RFPC0065 RTU
1557      t_ExpResult.CancelRange;                    //RFPC0065 RTU
```

```
1558      t_ExpResult.SetRange(['269030'],['269030']);    //RFPC0065 RTU
1559      t_ExpResult.First;                              //RFPC0065 RTU
1560      while not t_ExpResult.EOF do begin              //RFPC0065 RTU
1561        t_ExpResult.Edit;                             //RFPC0065 RTU
1562        t_ExpResult.FieldByName('Maintenance and Support').AsFloat := Value;     //RFPC0065 RTU
1563        if t_PropTax.FindKey(['269030'])
1564          then begin                                  //RFPC0065 RTU
1565                 if t_PropTax.FieldByName('Property Tax Flag').AsInteger = 1     //RFPC0065 RTU
1566                    then t_ExpResult.FieldByName('Property Tax').AsFloat :=      //RFPC0065 RTU
1567  //RFPC0065 RTU
1568                         t_PropTax.FieldByName('Property Tax').AsFloat           //RFPC0065 RTU
1569                    else t_ExpResult.FieldByName('Property Tax').AsString := ''; //RFPC0065 RTU
1570               end                                    //RFPC0065 RTU
1571          else t_ExpResult.FieldByName('Property Tax').AsString := '';          //RFPC0065 RTU
1572        t_ExpResult.Post;                             //RFPC0065 RTU
1573        t_ExpResult.Next;                             //RFPC0065 RTU
1574      end;
1575
1576      // RFPC 0006
1577      // cancel all ranges for the expense base table and result table,
1578      t_Expense.IndexName:='';
1579      t_Expense.CancelRange;
1580      t_ExpResult.IndexName:='';
1581      t_ExpResult.CancelRange;
1582      // Create the common cost worksheets
1583      CommonCostWorksheets(Market);
1584
1585
1586      // close all the tables
1587      t_Expense.Close;
1588      t_ExpLife.Close;
1589      t_ExpBase.Close;
1590      t_RoExpense.Close;
1591      t_ExpResult.Close;
1592      t_PropTax.Close;
1593    end;
1594
1595  function  Tf_Expense.PV(amount,rate, year: double) : double;
1596    begin
```

```
1597      // RFPC 0034 (changed from 99 to 0)
1598      if year = 0 then begin
1599        result := 0;
1600        exit;
1601      end;
1602      PV := amount/exp(ln(1.0+rate)*year);
1603    end;
1604
1605  function Tf_Expense.PMT(amount,rate,year: double): double;
1606    begin
1607     Result := amount*rate*exp(ln(1.0+rate)*year)/(exp(ln(1.0+rate)*year)-1.0);
1608    end;
1609
1610  function Tf_Expense.SFD(salvage,rate,year: double): double;
1611    begin
1612      // RFPC 0034 (changed from 99 to 0)
1613      if year = 0 then begin
1614        result := 0;
1615        exit;
1616      end;
1617      SFD := (1.0-salvage)*rate/(exp(ln(1.0+rate)*year)-1.0);
1618    end;
1619
1620  //RFPC0051
1621  function  Tf_Expense.DepreciationAndReturn (rate_return, life, salvage : Double): Double;
1622    begin
1623      // RFPC 0034, set the result to the rate of return if life is zero
1624      if (life = 0) then Result:=rate_return
1625      else Result := PMT(1.0-PV(salvage, rate_return, life), rate_return, life);
1626    end;
1627
1628  //RFPC0051
1629  function Tf_Expense.CompositeIncomeTax(rate_return, life, salvage : Double; macrs_yr : integer): Double;
1630    Var
1631      sfdep             : Double;
1632      sldep             : Double;
1633      debt_ratio        : Double;
1634      debt_rate         : Double;
1635      Composite_rate    : Double;
```

```
1636      Income_tax_Factor : Double;
1637      pvmacrs           : Double;
1638
1639    begin
1640      sfdep := SFD(salvage,rate_return,life);
1641
1642      // RFPC 0034
1643      if life = 0 then sldep := 0
1644      else if (life > 0) then sldep := (1.0-salvage)/life
1645      else
1646        begin
1647          sldep := 9999.9;
1648          MessageDlg(AccountStr+': Negative value entered for life.',mtWarning,[mbOK],0);
1649        end;
1650
1651      Composite_rate := t_ExpBase.FieldByName('State Rate').AsFloat +
1652                        t_ExpBase.FieldByName('Federal Rate').AsFloat *
1653                        (1.0 - t_ExpBase.FieldByName('State Rate').AsFloat);
1654
1655      debt_ratio := t_ExpBase.FieldByName('Long Term Fact').AsFloat +
1656                    t_ExpBase.FieldByName('Short Term Fact').AsFloat;
1657      if debt_ratio > 0.0 then
1658        debt_rate := ( t_ExpBase.FieldByName('Long Term Fact').AsFloat *
1659                       t_ExpBase.FieldByName('Long Term Rate').AsFloat +
1660                       t_ExpBase.FieldByName('Short Term Fact').AsFloat *
1661                       t_ExpBase.FieldByName('Short Term Rate').AsFloat ) / debt_ratio
1662      else
1663        begin
1664          debt_rate := 9999.9;
1665          MessageDlg(AccountStr+': Debt ratio is zero or negative.',mtWarning,[mbOK],0);
1666        end;
1667
1668      if ((1.0 - Composite_Rate)> 0.0) and (Rate_Return > 0.0) then
1669        income_tax_factor := (Composite_rate/(1-Composite_rate))*
1670                             (1-debt_ratio*debt_rate/rate_return)
1671      else
1672        begin
1673          income_tax_factor := 9999.9;
1674          if (Rate_Return <= 0.0) then
```

```
1675            MessageDlg(AccountStr+': Rate of return is zero or negative.',mtWarning,[mbOK],0)
1676          else MessageDlg(AccountStr+': (1-Composite Rate) is zero or negative.',mtWarning,[mbOK],0);
1677        end;
1678
1679      pvmacrs := PV_Macrs(macrs_yr, rate_return);
1680
1681      // RFPC 0034, set the result to ITF*Rate of Return if life is zero
1682      if (life = 0) then Result := income_tax_factor * rate_return
1683      else Result := income_tax_factor * (rate_return + sfdep - sldep)-
1684                     Composite_rate * (1+income_tax_factor)*
1685                     ( PMT(pvmacrs, rate_return, life) - sldep);
1686   end;
1687
1688  function Tf_Expense.PV_Macrs (macrs_yr : integer; rate : double): double;
1689   Const
1690     three   : array[1..4]  of Double =(0.3333, 0.4444, 0.1482, 0.0741);
1691     five    : array[1..6]  of Double =(0.2000, 0.3200, 0.1920, 0.1152, 0.1152, 0.0576);
1692     seven   : array[1..8]  of Double =(0.1429, 0.2450, 0.1750, 0.1250, 0.0892, 0.0892, 0.0891, 0.0446);
1693     ten     : array[1..11] of Double =(0.1000, 0.1800, 0.1440, 0.1152, 0.0922, 0.0737, 0.0655, 0.0655,
1694                                        0.0655, 0.0655, 0.0329);
1695     fifteen : array[1..16] of Double =(0.0500, 0.0950, 0.0855, 0.0770, 0.0693, 0.0623, 0.0590, 0.0590,
1696                                        0.0590, 0.0590, 0.0590, 0.0590, 0.0590, 0.0590, 0.0590, 0.0299);
1697     twenty  : array[1..21] of Double =(0.0375, 0.0722, 0.0668, 0.0618, 0.0571, 0.0529, 0.0489, 0.0452,
1698                                        0.0446, 0.0446, 0.0446, 0.0446, 0.0446, 0.0446, 0.0446, 0.0446,
1699                                        0.0446, 0.0446, 0.0446, 0.0446, 0.0224);
1700     thirty  : array[1..32] of Double =(0.0158, 0.0317, 0.0317, 0.0317, 0.0317, 0.0317, 0.0317, 0.0317,
1701                                        0.0317, 0.0317, 0.0317, 0.0317, 0.0317, 0.0317, 0.0317, 0.0317,
1702                                        0.0317, 0.0317, 0.0317, 0.0317, 0.0317, 0.0317, 0.0317, 0.0317,
1703                                        0.0317, 0.0317, 0.0317, 0.0317, 0.0317, 0.0317, 0.0317, 0.0332);
1704   Var
1705    i     : word;
1706    pvmacrs : Double;
1707
1708   begin
1709     pvmacrs := 0.0;
1710     case macrs_yr of
1711       3: for i := 1 to 4  do pvmacrs := pvmacrs + PV(three[i],rate,i);
1712       5: for i := 1 to 6  do pvmacrs := pvmacrs + PV(five[i],rate,i);
1713       7: for i := 1 to 8  do pvmacrs := pvmacrs + PV(seven[i],rate,i);
```

```
1714      10: for i := 1 to 11 do pvmacrs := pvmacrs + PV(ten[i],rate,i);
1715      15: for i := 1 to 16 do pvmacrs := pvmacrs + PV(fifteen[i],rate,i);
1716      20: for i := 1 to 21 do pvmacrs := pvmacrs + PV(twenty[i],rate,i);
1717      32: for i := 1 to 32 do pvmacrs := pvmacrs + PV(thirty[i],rate,i);
1718     end;
1719
1720    PV_Macrs := pvmacrs;
1721   end;
1722
1723  procedure Tf_Expense.SetCaption;
1724  begin
1725    Caption := 'Expenses [' + ExtractFilename(TabName)+']';
1726  end;
1727
1728  procedure Tf_Expense.FormResize(Sender: TObject);
1729  begin
1730  if ClientWidth < 350
1731      then ClientWidth := 350;
1732
1733   dbg_Expense.Width := ClientWidth-20;
1734   dbn_Expense.Width := 191;
1735   ts_Expense.Width := dbg_Expense.Width;
1736
1737   dbg_Expense.Left := 10;
1738   ts_Expense.Left := 10;
1739   if ClientWidth > (dbn_Expense.Width + cb_Assoc.Width + 5)
1740     then dbn_Expense.Left := (ClientWidth - (dbn_Expense.Width + cb_Assoc.Width + 5)) div 2
1741     else dbn_Expense.Left := 1;
1742   cb_Assoc.Left    := dbn_Expense.Left + dbn_Expense.Width + 5;
1743
1744   dbg_Expense.Top := l_CopyRight.Height+2;
1745   dbn_Expense.Top := ClientHeight - (dbn_Expense.Height + 5);
1746   cb_Assoc.Top :=  dbn_Expense.Top + (dbn_Expense.Height-cb_Assoc.Height) div 2;
1747   ts_Expense.Top := dbn_Expense.Top - (ts_Expense.Height+5);
1748
1749   dbg_Expense.Height :=  ts_Expense.Top - dbg_Expense.Top;
1750  end;
1751
1752  procedure Tf_Expense.FormShow(Sender: TObject);
```

```
1753  begin
1754    cb_Assoc.Checked := False;
1755    ts_Expense.TabIndex := 0;
1756    l_Copyright.Caption := f_associate.GetFileName(5);
1757
1758    TabName :=  f_associate.GetFileName(3002);
1759    if TabName = ''
1760      then exit;
1761
1762    With t_Expense do
1763      begin
1764         if Active then Close;
1765         DatabaseName := ExtractFilePath(TabName);
1766         TableName := ExtractFilename(TabName);
1767         TableType := ttDefault;
1768         Exclusive := True;
1769         Open;
1770      end;
1771    SetCaption;
1772  end;
1773
1774  procedure Tf_Expense.Exit1Click(Sender: TObject);
1775  begin
1776    if cb_Assoc.Checked
1777      then case ts_Expense.TabIndex of
1778            0 : f_associate.SetFileName(3002, TabName);
1779            1 : f_associate.SetFileName(3005, TabName);
1780            2 : f_associate.SetFileName(3006, TabName);
1781            3 : f_associate.SetFileName(3010, TabName);
1782           end;
1783    cb_Assoc.Checked := False;
1784    Hide;
1785  end;
1786
1787  procedure Tf_Expense.FormHide(Sender: TObject);
1788  begin
1789    if t_Expense.Active then t_Expense.close;
1790  end;
1791
```

```
1792  procedure Tf_Expense.New1Click(Sender: TObject);
1793  begin
1794    sd_Expense.Filter := 'Database|*.db';
1795
1796    if Not sd_Expense.execute then exit;
1797    TabName := sd_Expense.FileName;
1798    cb_Assoc.Checked := True;
1799
1800    case ts_Expense.TabIndex of
1801      0 : Make_Expense;
1802      1 : Make_ExpLife;
1803      2 : Make_ExpBase;
1804      3 : Make_PropTax;
1805    end;
1806
1807    SetCaption;
1808  end;
1809
1810  procedure Tf_Expense.Open1Click(Sender: TObject);
1811  begin
1812    od_Expense.Filter := 'Database|*.db;*.dbf*';
1813
1814    if Not od_Expense.execute then exit;
1815    TabName := od_Expense.FileName;
1816    With t_Expense do
1817      begin
1818         if Active then Close;
1819         DatabaseName := ExtractFilePath(TabName);
1820         TableName := ExtractFilename(TabName);
1821         TableType := ttDefault;
1822         Exclusive := True;
1823         Open;
1824      end;
1825    SetCaption;
1826  end;
1827
1828  procedure Tf_Expense.Import1Click(Sender: TObject);
1829  Var
1830    FTables : TextFile;
```

```
1831    Data     : String;
1832    Fld      : String;
1833    I        : word;
1834    index    : word;
1835
1836  begin
1837   t_Expense.Close;
1838   ImportDataCall('', AliasPath+'\'+t_Expense.Tablename, 10);
1839   t_Expense.Open
1840  end;
1841
1842  procedure Tf_Expense.Export1Click(Sender: TObject);
1843  begin
1844    ExportTable(t_Expense);
1845    WeAreWatching.LogEvent(mNORMAL,'Export1Click',' Expense Table ['+t_Expense.TableName+'] Exported.')
1846  end;
1847
1848   procedure Tf_Expense.Make_Expense;
1849   begin
1850    with t_Expense do
1851      begin
1852        Active := False;
1853        DatabaseName := ExtractFilePath(TabName);
1854        TableName := ExtractFilename(TabName);
1855        TableType := ttParadox;
1856        Exclusive := True;
1857        with FieldDefs do
1858          begin
1859            Clear;
1860            Add ('Account',      FTString, 6,  True);
1861            Add ('Category',     FTString, 1,  True);
1862            Add ('Amount',       FTFloat,  0,  False);
1863            Add ('Adjust1',      FTFloat,  0,  False);
1864            Add ('Adjust2',      FTFloat,  0,  False);
1865            Add ('Adjust3',      FTFloat,  0,  False);
1866            Add ('Default Adj3', FTFloat,  0,  False);  // RFPC0029
1867            Add ('Cable',        FTFloat,  0,  False);
1868            Add ('FiAer',        FTFloat,  0,  False);
1869            Add ('CuAer',        FTFloat,  0,  False);
```

```
1870            Add ('FiBur',         FTFloat,  0,  False);
1871            Add ('CuBur',         FTFloat,  0,  False);
1872            Add ('FiUnd',         FTFloat,  0,  False);
1873            Add ('CuUnd',         FTFloat,  0,  False);
1874            Add ('Poles',         FTFloat,  0,  False);
1875            Add ('Conduit',       FTFloat,  0,  False);
1876            Add ('Transmission', FTFloat,  0,  False);   // RFPC0037
1877            Add ('Switch',        FTFloat,  0,  False);
1878            Add ('Access',        FTFloat,  0,  False);
1879            Add ('IOT',           FTFloat,  0,  False);
1880            Add ('Direct Other', FTFloat,  0,  False);
1881            Add ('Bill_Pay',      FTFloat,  0,  False);
1882            Add ('Oper_Serv',     FTFloat,  0,  False);
1883            Add ('PubCom',        FTFloat,  0,  False);
1884            Add ('Consumer',      FTFloat,  0,  False);
1885            Add ('Business',      FTFloat,  0,  False);
1886            Add ('CXR',           FTFloat,  0,  False);
1887            Add ('Common',        FTFloat,  0,  False);
1888          end;
1889
1890        with IndexDefs do
1891          begin
1892            Clear;
1893            Add('Main', 'Account;Category', [ixPrimary, ixUnique]);
1894            Add('ByCat', 'Category;Account', [ixCaseInsensitive]);
1895          end;
1896
1897        CreateTable;
1898        Open;
1899      end;
1900  end;
1901
1902 procedure Tf_Expense.Make_ExpBase;
1903  begin
1904   with t_Expense do
1905     begin
1906       Active := False;
1907       DatabaseName := ExtractFilePath(TabName);
1908       TableName := ExtractFilename(TabName);
```

```
1909        TableType := ttParadox;
1910        Exclusive := True;
1911        with FieldDefs do
1912          begin
1913            Clear;
1914            Add ('Long Term Fact',  FTFloat,   0, True);
1915            Add ('Long Term Rate',  FTFloat,   0, True);
1916            Add ('Short Term Fact', FTFloat,   0, True);
1917            Add ('Short Term Rate', FTFloat,   0, True);
1918            Add ('Common Fact',     FTFloat,   0, True);
1919            Add ('Common Rate',     FTFloat,   0, True);
1920            Add ('PRFD Fact',       FTFloat,   0, True);
1921            Add ('PRFD Rate',       FTFloat,   0, True);
1922            Add ('Rate Return',     FTFloat,   0, True);
1923            Add ('State Rate',      FTFloat,   0, True);
1924            Add ('Federal Rate',    FTFloat,   0, True);
1925            Add ('Property Tax',    FTFloat,   0, True);
1926            Add ('Total Revenues',  FTFloat,   0, True);
1927            Add ('Data Year',       FTInteger, 0, True);
1928          end;
1929
1930        with IndexDefs do
1931          begin
1932            Clear;
1933          end;
1934
1935        CreateTable;
1936        Open;
1937      end;
1938  end;
1939
1940  procedure Tf_Expense.Make_PropTax;
1941   begin
1942    with t_Expense do
1943      begin
1944        Active := False;
1945        DatabaseName := ExtractFilePath(TabName);
1946        TableName := ExtractFilename(TabName);
1947        TableType := ttParadox;
```

```
1948        Exclusive := True;
1949        with FieldDefs do
1950          begin
1951            Clear;
1952            Add ('Account',           FTString,  6,  True);
1953            Add ('Property Tax Flag', FTInteger, 0,  True);
1954            Add ('Property Tax',      FTFloat,   0,  True);
1955          end;
1956
1957        with IndexDefs do
1958          begin
1959            Clear;
1960            Add('Main', 'Account', [ixPrimary, ixUnique]);
1961          end;
1962
1963        CreateTable;
1964        Open;
1965      end;
1966  end;
1967
1968  procedure Tf_Expense.Make_ExpLife;
1969   begin
1970    with t_Expense do
1971      begin
1972        Active := False;
1973        DatabaseName := ExtractFilePath(TabName);
1974        TableName := ExtractFilename(TabName);
1975        TableType := ttParadox;
1976        Exclusive := True;
1977        with FieldDefs do
1978          begin
1979            Clear;
1980            Add ('Account',       FTString, 6,  True);
1981            Add ('Economic Life', FTFloat, 0, True);
1982            Add ('Econ Salvage',  FTFloat, 0, True);
1983            Add ('Book Life',     FTFloat, 0, True);
1984            Add ('Book Salvage',  FTFloat, 0, True);
1985            Add ('Macrs Years',   FTFloat, 0, True);
1986          end;
```

```
1987
1988        with IndexDefs do
1989          begin
1990            Clear;
1991            Add('Main', 'Account', [ixPrimary, ixUnique]);
1992          end;
1993
1994        CreateTable;
1995        Open;
1996      end;
1997  end;
1998
1999  // RFPC 0006
2000  // create the table for the joint/shared costs
2001   procedure Tf_Expense.Make_JointShared;
2002   begin
2003    with t_JointSharedCost do
2004      begin
2005        Active := False;
2006        DatabaseName := 'Modata';
2007        TableName := 'ExpJointShared';
2008        TableType := ttParadox;
2009        Exclusive := True;
2010        with FieldDefs do
2011          begin
2012            Clear;
2013            Add ('Account',                  FTString,    32,  True);
2014            Add ('Category',                 FTString,     1,  True);
2015            Add ('Total',                    FTCurrency,   0,  False);
2016            Add ('Cable',                    FTFloat,      0,  False);
2017            Add ('Aerial Non-Metallic',      FTFloat,      0,  False);
2018            Add ('Aerial Metallic',          FTFloat,      0,  False);
2019            Add ('Buried Non-Metallic',      FTFloat,      0,  False);
2020            Add ('Buried Metallic',          FTFloat,      0,  False);
2021            Add ('Underground Non-Metallic', FTFloat,      0,  False);
2022            Add ('Underground Metallic',     FTFloat,      0,  False);
2023            Add ('Poles',                    FTFloat,      0,  False);
2024            Add ('Conduit',                  FTFloat,      0,  False);
2025            Add ('Transmission',             FTFloat,      0,  False);
```

```
2026            Add ('Switch',                  FTFloat,     0,  False);
2027            Add ('IOT',                     FTFloat,     0,  False);
2028            Add ('Direct Other',            FTFloat,     0,  False);
2029            Add ('Access',                  FTFloat,     0,  False);
2030            Add ('B and C',                 FTFloat,     0,  False);
2031            Add ('Oper',                    FTFloat,     0,  False);
2032            Add ('PUB COMM',                FTFloat,     0,  False);
2033            Add ('Consumer',                FTFloat,     0,  False);
2034            Add ('Business',                FTFloat,     0,  False);
2035            Add ('CXR',                     FTFloat,     0,  False);
2036            Add ('Common',                  FTFloat,     0,  False);
2037          end;
2038
2039        with IndexDefs do
2040          begin
2041            Clear;
2042            Add('Main', 'Account;Category', [ixPrimary, ixUnique]);
2043            Add('ByCat', 'Category;Account', [ixCaseInsensitive]);
2044          end;
2045
2046        CreateTable;
2047        IndexName:='ByCat';
2048        Open;
2049      end;
2050   end;
2051
2052  // RFPC 0006
2053  // create the table for the common costs worksheet
2054   procedure Tf_Expense.Make_Worksheets;
2055   begin
2056    with t_Worksheets do
2057      begin
2058        Active := False;
2059        DatabaseName := 'Modata';
2060        TableName := 'ExpCCWorksheet';
2061        TableType := ttParadox;
2062        Exclusive := True;
2063        with FieldDefs do
2064          begin
```

```
2065            Clear;
2066            Add ('Account',                 FTString,    32,  True);
2067            Add ('Category',                FTString,     1,  True);
2068            Add ('Amount',                  FTCurrency,   0,  False);
2069            Add ('Adjust1',                 FTFloat,      0,  False);
2070            Add ('Adjust2',                 FTFloat,      0,  False);
2071            Add ('Adjust3',                 FTFloat,      0,  False);
2072            Add ('New Amount',              FTCurrency,   0,  False);
2073            Add ('%BC',                     FTFloat,      0,  False);
2074            Add ('%Consumer',               FTFloat,      0,  False);
2075            Add ('%Business',               FTFloat,      0,  False);
2076            Add ('%CXR',                    FTFloat,      0,  False);
2077            Add ('%Common',                 FTFloat,      0,  False);
2078            Add ('$BC',                     FTCurrency,   0,  False);
2079            Add ('$Consumer',               FTCurrency,   0,  False);
2080            Add ('$Business',               FTCurrency,   0,  False);
2081            Add ('$CXR',                    FTCurrency,   0,  False);
2082            Add ('$Common',                 FTCurrency,   0,  False);
2083            Add ('SubTotal',                FTCurrency,   0,  False);
2084            Add ('ICM Based Factors',       FTFloat,      0,  False);
2085            Add ('Total',                   FTCurrency,   0,  False);
2086          end;
2087
2088        with IndexDefs do
2089          begin
2090            Clear;
2091            Add('Main', 'Account;Category', [ixPrimary, ixUnique]);
2092            Add('ByCat', 'Category;Account', [ixCaseInsensitive]);
2093          end;
2094
2095        CreateTable;
2096        IndexName:='ByCat';
2097        Open;
2098      end;
2099  end;
2100
2101  {Call generic print for expenses table}
2102  procedure Tf_Expense.PrintExpense( Open : Boolean);
2103  begin
```

```
2104      if Open then GenericPrintTable(t_Expense, Nil, 3002, True, False, 'Expenses')
2105      else GenericPrintTable(t_Expense, Nil, 3002, False, False, 'Expenses');
2106    end;
2107
2108    {Call generic print for Plant Life table}
2109    procedure Tf_Expense.PrintExpLife( Open : Boolean);
2110    begin
2111      if Open then GenericPrintTable(t_Expense, Nil, 3005, True, False, 'Plant Life')
2112      else GenericPrintTable(t_Expense, Nil, 3005, False, False, 'Plant Life');
2113    end;
2114
2115    {Call generic print for Expense Base table}
2116    procedure Tf_Expense.PrintExpBase( Open : Boolean);
2117    begin
2118      if Open then GenericPrintTable(t_Expense, Nil, 3006, True, False, 'Expense Base')
2119      else GenericPrintTable(t_Expense, Nil, 3006, False, False, 'Expense Base');
2120    end;
2121
2122    {Call generic print for Property Tax table}
2123    procedure Tf_Expense.PrintPropTax( Open : Boolean);
2124    begin
2125      if Open then GenericPrintTable(t_Expense, Nil, 3010, True, False, 'Property Tax')
2126      else GenericPrintTable(t_Expense, Nil, 3010, False, False, 'Property Tax');
2127    end;
2128
2129
2130    procedure Tf_Expense.Print1Click(Sender: TObject);
2131    begin
2132      case ts_Expense.TabIndex of
2133        0 : GenericPrintTable(t_Expense, Nil, 3002, False, True, 'Expenses');
2134        1 : GenericPrintTable(t_Expense, Nil, 3005, False, True, 'Plant Life');
2135        2 : GenericPrintTable(t_Expense, Nil, 3006, False, True, 'Expense Base');
2136        3 : GenericPrintTable(t_Expense, Nil, 3010, False, True, 'Property Tax');
2137      end;
2138    end;
2139
2140    procedure Tf_Expense.DoTabChange(NewTab: Integer);
2141    begin
2142      if t_Expense.Active then t_Expense.Close;
```

```
2143    t_Expense.IndexFieldNames := '';
2144
2145    case Newtab of
2146     0 : TabName :=  f_associate.GetFileName(3002);
2147     1 : TabName :=  f_associate.GetFileName(3005);
2148     2 : TabName :=  f_associate.GetFileName(3006);
2149     3 : TabName :=  f_associate.GetFileName(3010);
2150    end;
2151
2152    if TabName = ''
2153      then begin
2154             sd_Expense.Filter := 'Database|*.db';
2155
2156             if Not sd_Expense.execute then exit;
2157             TabName := sd_Expense.FileName;
2158             cb_Assoc.Checked := True;
2159
2160             case NewTab of
2161              0 : Make_Expense;
2162              1 : Make_ExpLife;
2163              2 : Make_ExpBase;
2164              3 : Make_PropTax;
2165             end;
2166           end
2167      else begin
2168             With t_Expense do
2169               begin
2170                 if Active then Close;
2171                 DatabaseName := ExtractFilePath(TabName);
2172                 TableName := ExtractFilename(TabName);
2173                 TableType := ttDefault;
2174                 Exclusive := True;
2175                 Open;
2176               end;
2177            end;
2178    SetCaption;
2179  end;
2180
2181  procedure Tf_Expense.ts_expenseChange(Sender: TObject; NewTab: Integer;
```

```
2182    var AllowChange: Boolean);
2183  begin
2184    if cb_Assoc.Checked
2185      then case ts_Expense.TabIndex of
2186            0 : f_associate.SetFileName(3002, TabName);
2187            1 : f_associate.SetFileName(3005, TabName);
2188            2 : f_associate.SetFileName(3006, TabName);
2189            3 : f_associate.SetFileName(3010, TabName);
2190           end;
2191    cb_Assoc.Checked := False;
2192
2193    DoTabChange (Newtab);
2194    if (NewTab = 2) and (t_expense.RecordCount = 1)
2195      then dbn_Expense.VisibleButtons := [nbEdit,nbPost,nbCancel,nbRefresh]
2196      else dbn_Expense.VisibleButtons := [nbFirst,nbPrior,nbNext,nbLast,nbInsert,
2197                                          nbDelete,nbEdit,nbPost,nbCancel,nbRefresh];
2198  end;
2199
2200  procedure Tf_Expense.HelplClick(Sender: TObject);
2201  begin
2202    Case ts_Expense.TabIndex of
2203      0 : Application.HelpCommand(HELP_CONTEXT, 430);
2204      1 : Application.HelpCommand(HELP_CONTEXT, 419);
2205      2 : Application.HelpCommand(HELP_CONTEXT, 420);
2206      3 : Application.HelpCommand(HELP_CONTEXT, 418);
2207    end;
2208  end;
2209
2210  procedure Tf_Expense.FormActivate(Sender: TObject);
2211  begin
2212    UnitName := 'EXPENSE';
2213  end;
2214
2215  // RFPC 0006
2216  // generate a report for the joint/shared costs
2217  procedure Tf_Expense.Report;
2218  Var
2219   totNonNetwork, GSA_OSP, GSA_Trans, GSA_Switch, GSA_IOT,
2220   NetAdm_OSP, NetAdm_Trans, NetAdm_Switch, NetAdm_IOT,
```

```
2221   GenAdm_OSP, GenAdm_Trans, GenAdm_Switch, GenAdm_IOT,
2222   OD_Network, OD_IOT, totShared  : Double;
2223   Ind : integer;
2224
2225   // compute all the ratios and shared costs
2226   procedure CalculateSharedCosts;
2227    var i:integer;
2228    begin
2229      totNonNetwork:=0;  GSA_OSP:=0;         GSA_Trans:=0;
2230      GSA_Switch:=0;     GSA_IOT:=0;         NetAdm_OSP:=0;
2231      NetAdm_Trans:=0;   NetAdm_Switch:=0;   NetAdm_IOT:=0;
2232      GenAdm_OSP:=0;     GenAdm_Trans:=0;    GenAdm_Switch:=0;
2233      GenAdm_IOT:=0;     OD_Network:=0;      OD_IOT:=0;    TotalIOT:=0;
2234
2235      totNonNetwork:=(SubTotal[16]+SubTotal[17]+SubTotal[18]+SubTotal[19]+
2236                      SubTotal[20]+SubTotal[21]+SubTotal[22]+SubTotal[23]);
2237      if (totNonNetwork < 0) then totNonNetwork:=0;
2238
2239      with t_JointSharedCost do
2240        begin
2241          // GSA
2242          CancelRange;
2243          SetRange(['6112'],['6124']);
2244          First;
2245          while not EOF do
2246            begin
2247              // add up all the values for cable through conduit
2248              for i:=3 to 11 do
2249                GSA_OSP:= GSA_OSP + Fields[i].AsFloat;
2250              GSA_Trans:=GSA_Trans+FieldByName('Transmission').AsFloat;
2251              GSA_Switch:=GSA_Switch+FieldByName('Switch').AsFloat;
2252              GSA_IOT:=GSA_IOT+FieldByName('IOT').AsFloat;
2253              Next;
2254            end;
2255          // Network Administration
2256          CancelRange;
2257          SetRange(['6512'],['6535']);
2258          First;
2259          while not EOF do
```

```
2260            begin
2261              // add up all the values for cable through conduit
2262              for i:=3 to 11 do
2263                NetAdm_OSP:= NetAdm_OSP + Fields[i].AsFloat;
2264              NetAdm_Trans:=NetAdm_Trans+FieldByName('Transmission').AsFloat;
2265              NetAdm_Switch:=NetAdm_Switch+FieldByName('Switch').AsFloat;
2266              NetAdm_IOT:=NetAdm_IOT+FieldByName('IOT').AsFloat;
2267              Next;
2268            end;
2269          // General Administration
2270          CancelRange;
2271          SetRange(['6611'],['7240']);
2272          First;
2273          while not EOF do
2274            begin
2275              // add up all the values for cable through conduit
2276              for i:=3 to 11 do
2277                GenAdm_OSP:= GenAdm_OSP + Fields[i].AsFloat;
2278              GenAdm_Trans:=GenAdm_Trans+FieldByName('Transmission').AsFloat;
2279              GenAdm_Switch:=GenAdm_Switch+FieldByName('Switch').AsFloat;
2280              GenAdm_IOT:=GenAdm_IOT+FieldByName('IOT').AsFloat;
2281              Next;
2282            end;
2283          // Network
2284          if (DirectExp[2] > 0) and (DirectExp[14] > 0) then
2285            OD_IOT := (DirectExp[14]/DirectExp[2])*SubTotal[15]
2286          else OD_IOT:=0;
2287          OD_Network:=SubTotal[15]-OD_IOT;
2288          totShared:=GSA_OSP+GSA_Trans+GSA_Switch+GSA_IOT+NetAdm_OSP+
2289                     NetAdm_Trans+NetAdm_Switch+NetAdm_IOT+GenAdm_OSP+
2290                     GenAdm_Trans+GenAdm_Switch+GenAdm_IOT+OD_Network+OD_IOT;
2291          TotalIOT:=GSA_IOT+NetAdm_IOT+GenAdm_IOT+OD_IOT;
2292        end; {with t_JointSharedCost}
2293    end; {CalculateSharedCosts}
2294
2295  begin
2296    t_RoExpense.Open;      // open Expense Run Time Options Table
2297    if (UpperCase(t_RoExpense.FieldByName('Market').AsString) = 'RETAIL') then
2298      Market:='Retail'
```

```
2299    else Market:='Wholesale';
2300    Category:=Copy(Market,1,1);
2301    ExpShared:=t_RoExpense.FieldByName('Shared').AsBoolean;
2302    t_RoExpense.Close;    // close Expense Run Time Options Table
2303
2304    // get the total revenue and the data year from the base table
2305    if not t_ExpBase.Active then t_ExpBase.Open;
2306    t_ExpBase.CancelRange;
2307    t_ExpBase.First;
2308    TotRev:=t_ExpBase.FieldByName('Total Revenues').AsFloat;
2309    DataYear:=t_ExpBase.FieldByName('Data Year').AsInteger;
2310    t_ExpBase.Close;
2311
2312    // All the information that is required comes from the t_JointSharedCost table.
2313    if not t_JointSharedCost.Active then t_JointSharedCost.Active:=true;
2314    t_JointSharedCost.IndexName:='';
2315    t_JointSharedCost.CancelRange;
2316    FillChar(SubTotal,SizeOf(ArrayDouble),0);
2317    FillChar(DirectExp,SizeOf(ArrayDouble),0);
2318    FillChar(TotInv,SizeOf(ArrayDouble),0);
2319    FillChar(Ratios,SizeOf(ArrayDouble),0);
2320
2321    // Get the total investment first
2322    if t_JointSharedCost.FindKey(['C) Total Investment']) then
2323      begin
2324        for Ind:=2 to 13 do TotInv[Ind]:=t_JointSharedCost.Fields[Ind].AsFloat;
2325      end;
2326    // Get the subtotals
2327    if t_JointSharedCost.FindKey(['A) Subtotal']) then
2328      begin
2329        for Ind:=2 to 23 do SubTotal[Ind]:=t_JointSharedCost.Fields[Ind].AsFloat;
2330      end;
2331    // Get the direct expenses
2332    if t_JointSharedCost.FindKey(['G) Direct Cost']) then
2333      begin
2334        for Ind:=2 to 14 do DirectExp[Ind]:=t_JointSharedCost.Fields[Ind].AsFloat;
2335      end;
2336    // Get the ratios
2337    if t_JointSharedCost.FindKey(['D) Ratios (B/C)']) then
```

```
2338      begin
2339        for Ind:=2 to 13 do Ratios[Ind]:=t_JointSharedCost.Fields[Ind].AsFloat;
2340      end;
2341
2342    CalculateSharedCosts;
2343    t_JointSharedCost.Close;
2344    // set main bookmark
2345    f_ReportView.AddBookMark(3, 0, 'Shared Cost');
2346    // print out the company study
2347    NetworkServicesReport;
2348    // print out the common cost summary
2349    SummaryReport;
2350  end; {report}
2351
2352  // RFPC 0006
2353  // Network Services Shared Cost page
2354  procedure Tf_Expense.NetworkServicesReport;
2355  var i : integer;
2356      Srch, Strg, PrtFmt, ActNo  : String;
2357      rTop, rBottom, rRight, rLeft, Std_ColSize : double;
2358
2359   // print header
2360   procedure PrintCompanyStudyHeader;
2361    Const Labels : array [1 .. 6] of string[20] =
2362       ('Account', 'OSP', 'Transmission', 'Switch', 'Network', 'Total');
2363    Var
2364      I    : Integer;
2365    begin
2366      Inc(Page);
2367      { Print Heading }
2368      PageObj.FntNam  := 'Arial';
2369      PageObj.FntSize := 14;
2370      PageObj.FntStyle:= [fsBold];
2371      PageObj.Color := clBlue;
2372
2373      // main title
2374      f_ReportView.PrintText(Page, 0.5,0.25, 6.0,0.5, 'Network Services Shared Cost',
2375  dt_Center+dt_SingleLine);
2376
```

```
2377      PageObj.FntSize := 10;
2378      PageObj.FntStyle:= [];
2379
2380      // state and market labels
2381      f_ReportView.PrintText(Page, 6.0,0.25, 8.0,0.45, 'State of '+f_associate.GetFileName(200),
2382  dt_Left+dt_SingleLine);
2383      f_ReportView.PrintText(Page, 6.0,0.45, 8.0,0.65, Market, dt_Left+dt_SingleLine);
2384
2385      // user defined header note
2386      PageObj.Color := clBlack;
2387      f_ReportView.PrintText(Page, 0.5, 0.85, 8.0,1.1, f_associate.GetFileName(201), dt_Left+dt_SingleLine);
2388
2389      // column headings
2390      PageObj.FntSize  := 8;
2391      PageObj.Color := clBlue;
2392      rTop := 1.1;
2393      rBottom := 1.45;
2394      f_ReportView.PrintBox(Page, 0.5, rTop, 8.0, rBottom, 1, clBlue, clWhite); //RFPC0006
2395      rLeft := 0.5;
2396      rRight := 8.0-(5*Std_Colsize);
2397      f_ReportView.PrintText(Page, rLeft,rTop+0.05, rRight,rBottom,
2398                             Labels[1], dt_Center+dt_WordBreak);
2399      for I := 2 to 6 do
2400        begin
2401          f_ReportView.PrintLine(Page, rRight,rTop,rRight,rBottom, 1);
2402          rLeft := rRight;
2403          rRight := rLeft + Std_Colsize;
2404          f_ReportView.PrintText(Page, rLeft,rTop+0.025, rRight,rBottom,
2405                             Labels[i], dt_Center+dt_WordBreak);
2406        end;
2407      rTop:=rBottom;
2408    end; {PrintCompanyStudyHeader}
2409
2410   // page footer
2411   procedure Footer(PageNum : Word);
2412    var TempStr : string;
2413    begin
2414      if ExpShared then TempStr:=f_associate.GetFileName(203)
2415      else TempStr:=f_associate.GetFileName(204);
```

```
2416
2417        PageObj.FntSize  := 10;
2418        PageObj.Color := clBlue;
2419        PageObj.FntStyle := [fsBold, fsItalic];
2420
2421        // proprietary notice
2422        Strg := f_associate.GetFileName(8);
2423        f_ReportView.PrintText(PageNum, 0.5,10.5, 8.0,10.7, Strg, dt_Center+dt_SingleLine);
2424
2425        PageObj.FntSize  := 8;
2426        PageObj.Color := clBlack;
2427        PageObj.FntStyle := [];
2428        // date time stamp
2429        Strg := FormatDateTime('mmmm dd, yyyy  hh:mm am/pm', Now);
2430        f_ReportView.PrintText(PageNum, 0.5,10.5, 2.75,10.7, Strg, dt_Left+dt_SingleLine);
2431
2432        // user defined footnote
2433        Strg := f_associate.GetFileName(202);
2434        if Strg <> ''
2435          then begin
2436                 f_ReportView.PrintText(PageNum, 0.5,10.3, 8.0,10.5, Strg, dt_Left+dt_SingleLine);
2437                 f_ReportView.PrintLine(PageNum, 0.5,10.2, 8.0, 10.2, 2);
2438                 PageObj.Color := clBlue;
2439                 PageObj.FntStyle:= [fsBold,fsItalic];
2440                 f_ReportView.PrintText(Page, 0.5, 10.0, 8.0,10.195, TempStr, dt_Left+dt_SingleLine);
2441                 PageObj.FntStyle:= [];
2442               end
2443          else begin
2444                 f_ReportView.PrintLine(PageNum, 0.5,10.4, 8.0, 10.4, 2);
2445                 PageObj.Color := clBlue;
2446                 PageObj.FntStyle:= [fsBold,fsItalic];
2447                 f_ReportView.PrintText(Page, 0.5, 10.2, 8.0,10.395, TempStr, dt_Left+dt_SingleLine);
2448                 PageObj.FntStyle:= [];
2449               end;
2450      end; {Footer}
2451
2452     // print the section specified by StartRange and EndRange
2453     procedure PrintCompanySection(SectionHeader, StartRange, EndRange    : string;
2454                                   showGrand, AddToMain, addZ: boolean);
```

```
2455    var i,j : integer;
2456        totOSP, totTransmission, totSwitch, totNetwork,
2457        Total, subOSP, totRow, TempNum : double;
2458        SecTotal : array [1 .. 5] of double;
2459        Account : string;
2460    begin
2461      // init the section totals
2462      FillChar(SecTotal,SizeOf(SecTotal),0);
2463      PageObj.FntSize  := 8;
2464      PageObj.Color := clBlue;
2465      PageObj.FntStyle := [fsBold];
2466      rLeft   := 0.5;
2467      rRight  := 8.0;
2468      rBottom := rTop+0.15;
2469      f_ReportView.PrintText(Page, rLeft, rTop+0.01, rRight, rBottom,
2470        StartRange+'-'+EndRange+'  '+SectionHeader, dt_Left+dt_SingleLine);
2471
2472        // set the appropriate range
2473      PageObj.Color := clBlack;
2474      PageObj.FntStyle := [];
2475      Total:=0;
2476      with t_JointSharedCost do
2477        begin
2478          CancelRange;    // cancel any previously defined ranges
2479          if addZ then SetRange(['Z'+StartRange],['Z'+EndRange])
2480          else SetRange([StartRange],[EndRange]);
2481          First;
2482          // cycle through all the records
2483          while not EOF do
2484            begin
2485              rLeft   := 1.0;
2486              rRight  := 8.0-(5*Std_ColSize);
2487              rTop:=rBottom;
2488              rBottom:=rTop+0.15;
2489              // print Account number first
2490              PageObj.FntStyle := [fsBold];
2491              if addZ then Account:=Copy(FieldByName('Account').AsString,2,4)
2492              else Account:=FieldByName('Account').AsString;
2493              f_ReportView.PrintText(Page, rLeft, rTop+0.02, rRight, rBottom,
```

```
2494              Account+'  '+GetActDesc(Account), dt_Left+dt_SingleLine);
2495
2496            PageObj.FntStyle := [];
2497            rLeft:=rRight;
2498            rRight:=rLeft+Std_Colsize;
2499            subOSP:=0;  totRow:=0;
2500            for j:=3 to 11 do
2501              subOSP:=subOSP+Fields[j].AsFloat;   // add up the OSP fields
2502            if AddToMain then
2503              GrandTotal[1]:=GrandTotal[1]+subOSP;  // add it to the OSP grand total
2504            SecTotal[1]:=SecTotal[1]+subOSP;      // add it to the section total
2505            Total:=Total+subOSP;                  // add it to main total
2506            totRow:=totRow+subOSP;                // add it to the current row total
2507            f_ReportView.PrintText(Page, rLeft, rTop+0.02, rRight, rBottom,
2508              Format('%12.0m',[subOSP]), dt_Right+dt_SingleLine);
2509            rLeft:=rRight;
2510            rRight:=rLeft+Std_Colsize;
2511            for i:=2 to 4 do
2512              begin
2513                if (i=2) then TempNum:=FieldByName('Transmission').AsFloat
2514                else if (i=3) then TempNum:=FieldByName('Switch').AsFloat
2515                //  RFPC 0033 make sure that the IOT portion is not included in the network costs
2516                else if (i=4) and (SubTotalCable_IOT <> 0) then
2517                  TempNum:=FieldByName('Direct Other').AsFloat*(1-(SubTotal[14]/SubTotalCable_IOT))
2518                else TempNum:=0;
2519                if AddToMain then
2520                  GrandTotal[i]:=GrandTotal[i]+TempNum;  // add it to the OSP grand total
2521                Total:=Total+TempNum;                  // add it to main total
2522                SecTotal[i]:=SecTotal[i]+TempNum;      // add it to the section total
2523                totRow:=totRow+TempNum;                // add it to the current row total
2524                f_ReportView.PrintText(Page, rLeft, rTop+0.02, rRight, rBottom,
2525                  Format('%12.0m',[TempNum]), dt_Right+dt_SingleLine);
2526                rLeft:=rRight;
2527                rRight:=rLeft+Std_Colsize;
2528              end; {for i}
2529            f_ReportView.PrintText(Page, rLeft, rTop+0.02, rRight, rBottom,
2530              Format('%12.0m',[totRow]), dt_Right+dt_SingleLine);
2531            Next;
2532          end; {while not eof}
```

```
2533        end; {with t_JointSharedCost}
2534      rLeft   := 1.0;
2535      rRight  := 8.0-(5*Std_ColSize);
2536      rTop:=rBottom+0.015;
2537      rBottom:=rTop+0.15;
2538      // print Account number first
2539      PageObj.FntStyle := [fsBold];
2540      f_ReportView.PrintText(Page, rLeft, rTop+0.02, rRight, rBottom,
2541         'Subtotal', dt_Left+dt_SingleLine);
2542      rLeft:=rRight;
2543      rRight:=rLeft+Std_Colsize;
2544      // print out the section totals
2545      for i:=1 to 4 do SecTotal[5]:=SecTotal[5]+SecTotal[i];
2546      for i:=1 to 5 do
2547        begin
2548          f_ReportView.PrintText(Page, rLeft, rTop+0.02, rRight, rBottom,
2549             Format('%12.0m',[SecTotal[i]]), dt_Right+dt_SingleLine);
2550          rLeft:=rRight;
2551          rRight:=rLeft+Std_Colsize;
2552        end;
2553
2554      PageObj.FntStyle := [];
2555      PageObj.Color := clBlack;
2556      rTop:=rBottom+0.025;
2557
2558      if not showGrand then exit;
2559      // show the grand total, followed by the IOT totals
2560      rLeft   := 0.5;
2561      rRight  := 8.0-(5*Std_ColSize);
2562      rTop:=rBottom+0.025;
2563      rBottom:=rTop+0.15;
2564      PageObj.Color := clBlue;
2565      // print row label first
2566      PageObj.FntStyle := [fsBold];
2567      f_ReportView.PrintText(Page, rLeft, rTop+0.02, rRight, rBottom,
2568         'Total', dt_Left+dt_SingleLine);
2569      rLeft:=rRight;
2570      rRight:=rLeft+Std_Colsize;
2571      for i:=1 to 4 do GrandTotal[5]:=GrandTotal[5]+GrandTotal[i];
```

```
2572      for i:=1 to 5 do
2573        begin
2574          f_ReportView.PrintText(Page, rLeft, rTop+0.02, rRight, rBottom,
2575             Format('%12.0m',[GrandTotal[i]]), dt_Right+dt_SingleLine);
2576          rLeft:=rRight;
2577          rRight:=rLeft+Std_Colsize;
2578        end;
2579
2580      rTop:=rBottom+0.025;
2581      PageObj.FntStyle := [];
2582      PageObj.Color := clBlack;
2583    end; {PrintCompanySection}
2584
2585  begin
2586    // open up the joint shared cost table, reset any previously defined ranges
2587    if not t_JointSharedCost.Active then t_JointSharedCost.Active:=true;
2588    t_JointSharedCost.IndexName:='';
2589    t_JointSharedCost.CancelRange;
2590    Std_ColSize := 6.5 / 8;
2591    // print out header and footer
2592    PrintCompanyStudyHeader;
2593    Footer(Page);
2594    // print out the network services page
2595    FillChar(GrandTotal,SizeOf(GrandTotal),0);
2596    PrintCompanySection('General Support Costs','2111','2124', false, true, false);
2597    PrintCompanySection('Plant Specific Operations Costs','6112','6124', false, true, false(true));
2598    PrintCompanySection('Plant Non-Specific Operations Costs','6512','6535', false, true, false);
2599    PrintCompanySection('Customer Operations Costs','6611','6623', false, true, false);
2600    PrintCompanySection('Corporate Operations Costs','6711','7240', true, true, false);
2601    // bookmark
2602    f_ReportView.AddBookMark(0, Page, 'Network Services Shared Cost');
2603    t_JointSharedCost.Close;
2604  end; {NetworkServicesReport}
2605
2606  // RFPC 0006
2607  // Summary report consists of two types:
2608  // 1) Common Cost Summary
2609  // 2) Comparison Price Fixed Allocator
2610  procedure Tf_Expense.SummaryReport;
```

```
2611  var i, sPage, cPage, wPage1, wPage2 : integer;
2612      Srch, Strg, PrtFmt, ActNo  : String;
2613      rTop, rBottom, rRight, rLeft, Std_ColSize : double;
2614      perCorporate, perGeneral, totCorporate, totGeneral : double;
2615   // Generic header for all four pages
2616   procedure MainHeader(PageNum:integer; Title : string);
2617    begin
2618      // set font and style
2619      PageObj.FntNam  := 'Arial';
2620      PageObj.FntSize := 14;
2621      PageObj.FntStyle:= [fsBold];
2622      PageObj.Color := clBlue;
2623
2624      // center the title according to the page orientation
2625      if (PageNum < 0) then
2626        f_ReportView.PrintText(PageNum, 0.5,0.25, 8.0,0.5, Title, dt_Center+dt_SingleLine)
2627      else f_ReportView.PrintText(PageNum, 0.5,0.25, 6.0,0.5, Title, dt_Center+dt_SingleLine);
2628
2629      PageObj.FntSize := 10;
2630      PageObj.FntStyle:= [];
2631
2632      // state and market labels
2633      if (PageNum < 0) then
2634        begin
2635          f_ReportView.PrintText(PageNum, 8.0,0.25, 10.0,0.45, 'State of '+f_associate.GetFileName(200),
2636  dt_Left+dt_SingleLine);
2637          f_ReportView.PrintText(PageNum, 8.0,0.45, 10.0,0.65, Market, dt_Left+dt_SingleLine);
2638        end
2639      else
2640        begin
2641          f_ReportView.PrintText(PageNum, 6.0,0.25, 8.0,0.45, 'State of '+f_associate.GetFileName(200),
2642  dt_Left+dt_SingleLine);
2643          f_ReportView.PrintText(PageNum, 6.0,0.45, 8.0,0.65, Market, dt_Left+dt_SingleLine);
2644        end;
2645
2646      // User defined header
2647      PageObj.Color := clBlack;
2648      PageObj.FntStyle:= [];
2649      if (PageNum < 0) then
```

```
2650        f_ReportView.PrintText(PageNum, 0.5, 0.85, 10.5,1.1, f_associate.GetFileName(201),
2651  dt_Left+dt_SingleLine)
2652      else f_ReportView.PrintText(PageNum, 0.5, 0.85, 8.0,1.1, f_associate.GetFileName(201),
2653  dt_Left+dt_SingleLine);
2654    end; {mainheader}
2655
2656   // page footer
2657   procedure Footer(PageNum:integer);
2658    begin
2659      // set font and style
2660      PageObj.FntSize  := 10;
2661      PageObj.Color := clBlue;
2662      PageObj.FntStyle := [fsBold, fsItalic];
2663
2664      // proprietary notice
2665      Strg := f_associate.GetFileName(8);
2666      if (PageNum < 0) then
2667        f_ReportView.PrintText(PageNum, 0.5,8.1, 10.5,8.3, Strg, dt_Center+dt_SingleLine)
2668      else f_ReportView.PrintText(PageNum, 0.5,10.5, 8.0,10.7, Strg, dt_Center+dt_SingleLine);
2669
2670      PageObj.FntSize  := 8;
2671      PageObj.Color := clBlack;
2672      PageObj.FntStyle := [];
2673      // date and time stamp
2674      Strg := FormatDateTime('mmmm dd, yyyy  hh:mm am/pm', Now);
2675      if (PageNum < 0) then
2676        f_ReportView.PrintText(PageNum, 0.5,8.1, 2.75,8.3, Strg, dt_Left+dt_SingleLine)
2677      else f_ReportView.PrintText(PageNum, 0.5,10.5, 2.75,10.7, Strg, dt_Left+dt_SingleLine);
2678
2679
2680      // user defined footnote
2681      Strg := f_associate.GetFileName(202);
2682      if Strg <> ''
2683        then begin
2684               if (PageNum < 0) then
2685                 begin
2686                   f_ReportView.PrintText(PageNum, 0.5,8.1, 10.5,8.3, Strg, dt_Left+dt_SingleLine);
2687                   f_ReportView.PrintLine(PageNum, 0.5,8.0, 10.5, 8.0, 2);
2688                 end
```

```
2689              else
2690                begin
2691                  f_ReportView.PrintText(PageNum, 0.5,10.3, 8.0,10.5, Strg, dt_Left+dt_SingleLine);
2692                  f_ReportView.PrintLine(PageNum, 0.5,10.2, 8.0, 10.2, 2);
2693                end;
2694            end
2695        else
2696          begin
2697            if (PageNum < 0) then
2698              f_ReportView.PrintLine(PageNum, 0.5,8.0, 10.5, 8.0, 2)
2699            else f_ReportView.PrintLine(PageNum, 0.5,10.4, 8.0, 10.4, 2);
2700          end;
2701    end; {Footer}
2702
2703    // Looksup the total for the specified account (ExpArray for expense accounts,
2704    // and InvArray for investment accounts.
2705    function GetValue(Act : string):double;
2706     var i : integer;
2707         found : boolean;
2708         TempNum : double;
2709     begin
2710       found:=false;   TempNum:=0;   i:=1;
2711       if ((Act[1]='6') or (Act[1]='7')) and (NumExpActs > 0) then  // expense
2712         begin
2713           while (i <= NumExpActs) and not found do
2714             if (ExpArray[i].Act=Act) then
2715               found:=true
2716             else i:=i+1;
2717           if found then TempNum:=ExpArray[i].Total;
2718         end
2719       else if (Act[1]='2') and (NumInvActs > 0) then  // investment
2720         begin
2721           while (i <= NumInvActs) and not found do
2722             if (InvArray[i].Act=Act) then
2723               found:=true
2724             else i:=i+1;
2725           if found then TempNum:=InvArray[i].Total;
2726         end;
2727       Result:=TempNum;
```

```
2728    end; {GetValue}
2729
2730   procedure CommonCostHeader(FirstTime : boolean);
2731    begin
2732      PageObj.FntSize  := 8;
2733      PageObj.Color := clBlack;
2734      rTop := 1.3;
2735      rBottom := 1.65;
2736      // draw the border
2737      f_ReportView.PrintBox(sPage, 0.5, rTop, 8.0, 10.0, 2, clBlack, clWhite);
2738      f_ReportView.PrintBox(sPage, 0.5, rTop, 8.0-Std_ColSize, 9.6, 1, clBlack, clWhite);
2739      f_ReportView.PrintLine(sPage, 0.5, rBottom, 8.0, rBottom, 2);
2740      if FirstTime then f_ReportView.PrintLine(sPage, 0.5, 4.1, 8.0, 4.1, 2);
2741      f_ReportView.PrintLine(sPage, 0.5+Std_ColSize, rTop, 0.5+Std_ColSize, 9.6, 1);
2742      rLeft := 0.5;
2743      rRight := rLeft+Std_ColSize;
2744      PageObj.FntStyle := [fsBold];
2745      // column labels
2746      f_ReportView.PrintText(sPage, rLeft, rTop+0.02, rRight,rBottom,
2747            'USOA Account', dt_Center+dt_WordBreak);
2748      f_ReportView.PrintText(sPage, 0.5+(2*Std_ColSize), rTop+0.02, 8.0-(2*Std_ColSize), rBottom,
2749            'Cost Categories', dt_Center+dt_WordBreak);
2750      f_ReportView.PrintText(sPage, 8.0-(2*Std_ColSize), rTop+0.02, 8.0-Std_ColSize, rBottom,
2751            'Costs $K', dt_Right+dt_WordBreak);
2752      f_ReportView.PrintText(sPage, 8.0-Std_ColSize, rTop+0.02, 8.0, rBottom,
2753            'Percent of Revenues', dt_Center+dt_WordBreak);
2754    end;
2755
2756   procedure CheckForSpace(PosOnPage : double);
2757    begin
2758      // check to see if there is enough space on the page
2759      if (PosOnPage > 9.4) then
2760        begin
2761          // spawn a new page
2762          Inc(Page);  // summary page
2763          sPage:=Page;
2764          MainHeader(sPage,'Summary of Common Costs (Cont)');
2765          Footer(sPage);
2766          CommonCostHeader(false);
```

```
2767          rTop:=rBottom;
2768          rBottom:=rTop+0.15;
2769        end;
2770    end;
2771
2772    // Print out the detail for the specified accounts on the Summary page.
2773    // Computes a subtotal that is passed back to the caller.
2774    procedure PrintDetail(Accounts : array of string;  var SubTotal : double);
2775     var I : integer;
2776         TempStr : string;
2777         TempNum : double;
2778     begin
2779       PageObj.FntSize  := 8;
2780       PageObj.Color := clBlack;
2781       SubTotal:=0;
2782       // cycle through all the specified accounts
2783       for i:=Low(Accounts) to High(Accounts) do
2784        if (Pos('X',Accounts[i]) > 0) or (GetValue(Accounts[i]) <> 0) then
2785         begin
2786           // check to see if there is enough space on the page
2787           rLeft:=0.5;
2788           rRight:=rLeft+Std_ColSize;
2789           if (Pos('X',Accounts[i]) > 0) then PageObj.FntStyle := [fsBold]
2790           else PageObj.FntStyle := [];
2791           f_ReportView.PrintText(sPage, rLeft,rTop+0.02, rRight-0.02,rBottom,
2792                                  Accounts[i], dt_Right+dt_WordBreak);
2793           PageObj.FntStyle := [];
2794
2795           if (Pos('X',Accounts[i]) > 0) then rLeft:=rRight
2796           else rLeft:=rRight+0.1;
2797           rRight:=8.0-(2*Std_ColSize);
2798           f_ReportView.PrintText(sPage, rLeft+0.05,rTop+0.02, rRight,rBottom,
2799                                  GetActDesc(Accounts[i]), dt_Left+dt_WordBreak);
2800           rLeft:=8.0-(2*Std_ColSize);
2801           rRight:=rLeft+Std_ColSize;
2802           if (Pos('X',Accounts[i]) > 0) then TempStr:=''
2803           else
2804             begin
2805               TempNum:=GetValue(Accounts[i]);
```

```
2806              SubTotal:=SubTotal+TempNum;
2807              TempNum:=Round(TempNum/1000);
2808              TempStr:=Format('%12.0m',[TempNum]);
2809            end;
2810          f_ReportView.PrintText(sPage, rLeft,rTop+0.02, rRight-0.05,rBottom,
2811                                 TempStr, dt_Right+dt_WordBreak);
2812          rTop:=rBottom;
2813          rBottom:=rTop+0.15;
2814          CheckForSpace(rBottom);
2815        end;
2816      PageObj.FntStyle := [];
2817    end; {PrintDetail}
2818
2819   procedure PrintSection(Accounts : array of string; Titlel,Title2 : string);
2820    var SubTotal : double;
2821    begin
2822      rTop:=rBottom+0.05;
2823      rBottom:=rTop+0.15;
2824      CheckForSpace(rBottom);
2825      f_ReportView.PrintText(sPage, 0.55+Std_ColSize, rTop+0.02, 8.0-(2*Std_ColSize), rBottom,
2826            Titlel, dt_Center+dt_WordBreak);
2827      rTop:=rBottom;
2828      rBottom:=rTop+0.15;
2829      CheckForSpace(rBottom);
2830      PrintDetail(Accounts, SubTotal);
2831      totGeneral:=totGeneral+SubTotal;
2832      SubTotal:=Round(SubTotal/1000);
2833      PageObj.FntStyle := [fsBold];
2834      CheckForSpace(rBottom);
2835      f_ReportView.PrintLine(sPage, 8.0-(2*Std_ColSize), rTop, 8.0-Std_ColSize, rTop, 1);
2836      f_ReportView.PrintText(sPage, 0.55+Std_ColSize, rTop+0.02, 8.0-(2*Std_ColSize), rBottom,
2837            Title2, dt_Left+dt_WordBreak);
2838      f_ReportView.PrintText(sPage, 8.0-(2*Std_ColSize), rTop+0.02, 8.0-Std_ColSize-0.05, rBottom,
2839            Format('%12.0m',[SubTotal]), dt_Right+dt_WordBreak);
2840    end;
2841
2842   // Print out the entire Summary page
2843   procedure SummaryDetail;
2844    var SubTotal, TempNum : double;
```

```
2845    begin
2846      CommonCostHeader(true);
2847      // total regulated revenues
2848      f_ReportView.PrintText(sPage, 0.55+Std_ColSize, 1.1, 8.0-(2*Std_ColSize), 1.3,
2849            IntToStr(DataYear)+' Total Regulated Revenues', dt_Left+dt_WordBreak);
2850      TempNum:=Round(TotRev/1000);
2851      f_ReportView.PrintText(sPage, 8.0-(2*Std_ColSize), 1.1, 8.0-Std_ColSize-0.05, 1.3,
2852            Format('%12.0m',[TempNum])+'K', dt_Right+dt_WordBreak);
2853
2854      // print out the Corporate operations costs
2855      totGeneral:=0;
2856      rTop:=rBottom;
2857      rBottom:=rTop+0.2;
2858      f_ReportView.PrintText(sPage, 0.55+Std_ColSize, rTop+0.02, 8.0-(2*Std_ColSize), rBottom,
2859            'I.  Corporate Operations Costs', dt_Left+dt_WordBreak);
2860      rTop:=rBottom;
2861      rBottom:=rTop+0.15;
2862
2863  PrintDetail(['671X','6711','6712','672X','6721','6722','6723','6724','6725','6726','6727','6728','7240'],
2864  SubTotal);
2865      totCorporate:=SubTotal;
2866      perCorporate:=(SubTotal/TotRev)*100;
2867      SubTotal:=Round(SubTotal/1000);
2868      PageObj.FntStyle := [fsBold];
2869      f_ReportView.PrintLine(sPage, 8.0-(2*Std_ColSize), rTop, 8.0-Std_ColSize, rTop, 1);
2870      f_ReportView.PrintText(sPage, 0.55+Std_ColSize, rTop+0.02, 8.0-(2*Std_ColSize), rBottom,
2871            'TOTAL 671X, 672X and 7240 COSTS', dt_Left+dt_WordBreak);
2872      f_ReportView.PrintText(sPage, 8.0-(2*Std_ColSize), rTop+0.02, 8.0-Std_ColSize-0.05, rBottom,
2873            Format('%12.0m',[SubTotal]), dt_Right+dt_WordBreak);
2874      rTop:=3.9;
2875      rBottom:=rTop+0.15;
2876      f_ReportView.PrintText(sPage, 0.55+Std_ColSize, rTop+0.02, 8.0-(2*Std_ColSize), rBottom,
2877            'TOTAL CORPORATE OPERATIONS COSTS', dt_Left+dt_WordBreak);
2878      f_ReportView.PrintText(sPage, 8.0-(2*Std_ColSize), rTop+0.02, 8.0-Std_ColSize-0.05, rBottom,
2879            Format('%12.0m',[SubTotal]), dt_Right+dt_WordBreak);
2880      f_ReportView.PrintText(sPage, 8.0-Std_ColSize, rTop+0.02, 8.0, rBottom,
2881            Format('%5.2f',[perCorporate])+'%', dt_Center+dt_WordBreak);
2882
2883      PageObj.FntStyle := [fsBold];
```

```
2884      // print out the general support and plant operating costs
2885      rTop:=4.1;
2886      rBottom:=rTop+0.2;
2887      f_ReportView.PrintText(sPage, 0.55+Std_ColSize, rTop+0.02, 8.0-(2*Std_ColSize), rBottom,
2888            'II.  GENERAL SUPPORT AND PLANT OPERATIONS COSTS', dt_Left+dt_WordBreak);
2889
2890      //RFPC 0038 and 0039
2891      PrintSection(['2111','2112','2113','2114','2115','2116','2121','2122','2123','2124'],
2892                   'General Support Costs','TOTAL GENERAL SUPPORT COSTS');
2893
2894      PrintSection(['611X','6112','6113','6114','6115','6116','612X','6121','6122','6123','6124',
2895                    '6211','6212','6215','6220','6231','6232','6311','6341','6351','6362',
2896                    '6411','6421','6422','6423','6424','6425','6426','6431','6441'],
2897                   'Plant Specific Operations Costs','TOTAL PLANT SPECIFIC OPERATIONS COSTS');
2898
2899      PrintSection(['6512','653X','6531','6532','6533','6534','6535'],
2900                   'Plant Non-Specific Operations Costs','TOTAL PLANT NON-SPECIFIC OPERATIONS COSTS');
2901
2902      //RFPC 0038 and 0039
2903      PrintSection(['66XX','6611','6612','6613','6621','6622','6623'],
2904                    'Customer Operations Costs','TOTAL CUSTOMER OPERATIONS COSTS');
2905
2906      rTop:=9.4;
2907      rBottom:=rTop+0.15;
2908      perGeneral:=(totGeneral/TotRev)*100;
2909      TempNum:=Round(totGeneral/1000);
2910      f_ReportView.PrintText(sPage, 0.55+Std_ColSize, rTop+0.02, 8.0-(2*Std_ColSize), rBottom,
2911            'TOTAL GENERAL SUPPORT AND PLANT OPERATIONS COSTS', dt_Left+dt_WordBreak);
2912      f_ReportView.PrintText(sPage, 8.0-(2*Std_ColSize), rTop+0.02, 8.0-Std_ColSize-0.05, rBottom,
2913            Format('%12.0m',[TempNum]), dt_Right+dt_WordBreak);
2914      f_ReportView.PrintText(sPage, 8.0-Std_ColSize, rTop+0.02, 8.0, rBottom,
2915            Format('%5.2f',[perGeneral])+'%', dt_Center+dt_WordBreak);
2916
2917      rTop:=9.65;
2918      rBottom:=rTop+0.15;
2919      TempNum:=Round((totGeneral+totCorporate)/1000);
2920      f_ReportView.PrintText(sPage, 0.55+Std_ColSize, rTop+0.02, 8.0-(2*Std_ColSize), rBottom,
2921            'TOTAL '+UpperCase(Market)+' COMMON COSTS', dt_Left+dt_WordBreak);
2922      f_ReportView.PrintText(sPage, 8.0-(2*Std_ColSize), rTop+0.02, 8.0-Std_ColSize-0.05, rBottom,
```

```
2923            Format('%12.0m',[TempNum]), dt_Right+dt_WordBreak);
2924
2925        TempNum:=((totGeneral+totCorporate)/TotRev)*100;
2926        f_ReportView.PrintText(sPage, 8.0-Std_ColSize, rTop+0.02, 8.0, rBottom,
2927            Format('%5.2f',[TempNum])+'%', dt_Center+dt_WordBreak);
2928        PageObj.FntStyle := [];
2929      end; {SummaryDetail}
2930
2931    // print out the Comparison Price Fixed Allocator page
2932    procedure ComparisonDetail;
2933      var TempNum : double;
2934      begin
2935        PageObj.FntSize  := 12;
2936        PageObj.Color := clBlack;
2937        PageObj.FntStyle := [fsBold];
2938        rTop:=1.1;
2939        rBottom:=rTop+0.25;
2940        TempNum:=Round(TotRev/1000);
2941        f_ReportView.PrintText(cPage, 0.55, rTop+0.02, 8.0-(2*Std_ColSize), rBottom,
2942            '(A) '+IntToStr(DataYear)+' Total Regulated Revenues', dt_Left+dt_WordBreak);
2943        f_ReportView.PrintText(cPage, 8.0-(2*Std_ColSize), rTop+0.02, 8.0, rBottom,
2944            Format('%12.0m',[TempNum])+'K', dt_Right+dt_WordBreak);
2945
2946        rTop:=rBottom+0.2;
2947        rBottom:=rTop+0.25;
2948        PageObj.FntStyle := [fsBold,fsUnderline];
2949        f_ReportView.PrintText(cPage, 0.55, rTop+0.02, 8.0-(2*Std_ColSize), rBottom,
2950            'Common Costs', dt_Left+dt_WordBreak);
2951
2952        rTop:=rBottom;
2953        rBottom:=rTop+0.25;
2954        PageObj.FntStyle := [fsBold];
2955        TempNum:=Round(totCorporate/1000);
2956        f_ReportView.PrintText(cPage, 0.55, rTop+0.02, 8.0-(2*Std_ColSize), rBottom,
2957            '(B) Corporate Operations Costs', dt_Left+dt_WordBreak);
2958        f_ReportView.PrintText(cPage, 8.0-(2*Std_ColSize), rTop+0.02, 8.0, rBottom,
2959            Format('%12.0m',[TempNum])+'K', dt_Right+dt_WordBreak);
2960
2961        rTop:=rBottom;
```

```
2962        rBottom:=rTop+0.25;
2963        TempNum:=Round(totGeneral/1000);
2964        f_ReportView.PrintText(cPage, 0.55, rTop+0.02, 8.0-(2*Std_ColSize), rBottom,
2965              '(C) General Support and Plant Operations Costs', dt_Left+dt_WordBreak);
2966        f_ReportView.PrintText(cPage, 8.0-(2*Std_ColSize), rTop+0.02, 8.0, rBottom,
2967              Format('%12.0m',[TempNum])+'K', dt_Right+dt_WordBreak);
2968
2969        rTop:=rBottom;
2970        rBottom:=rTop+0.25;
2971        TempNum:=Round((totGeneral+totCorporate)/1000);
2972        f_ReportView.PrintText(cPage, 0.55, rTop+0.02, 8.0-(2*Std_ColSize), rBottom,
2973              '(D) Total Common Costs', dt_Left+dt_WordBreak);
2974        f_ReportView.PrintText(cPage, 8.0-(2*Std_ColSize), rTop+0.02, 8.0, rBottom,
2975              Format('%12.0m',[TempNum])+'K', dt_Right+dt_WordBreak);
2976
2977        rTop:=rBottom+0.2;
2978        rBottom:=rTop+0.25;
2979        PageObj.FntStyle := [fsBold,fsUnderline];
2980        f_ReportView.PrintText(cPage, 0.55, rTop+0.02, 8.0-(2*Std_ColSize), rBottom,
2981              'Common Costs as a % of Revenues', dt_Left+dt_WordBreak);
2982
2983        rTop:=rBottom;
2984        rBottom:=rTop+0.25;
2985        PageObj.FntStyle := [fsBold];
2986        f_ReportView.PrintText(cPage, 0.55, rTop+0.02, 8.0-(2*Std_ColSize), rBottom,
2987              '(E) Corporate Operations Costs (B/A)', dt_Left+dt_WordBreak);
2988        f_ReportView.PrintText(cPage, 8.0-(2*Std_ColSize), rTop+0.02, 8.0, rBottom,
2989              Format('%5.2f',[perCorporate])+'%', dt_Right+dt_WordBreak);
2990
2991        rTop:=rBottom;
2992        rBottom:=rTop+0.25;
2993        f_ReportView.PrintText(cPage, 0.55, rTop+0.02, 8.0-(2*Std_ColSize), rBottom,
2994              '(F) General Support and Plant Operations Costs (C/A)', dt_Left+dt_WordBreak);
2995        f_ReportView.PrintText(cPage, 8.0-(2*Std_ColSize), rTop+0.02, 8.0, rBottom,
2996              Format('%5.2f',[perGeneral])+'%', dt_Right+dt_WordBreak);
2997
2998        rTop:=rBottom;
2999        rBottom:=rTop+0.25;
3000        TempNum:=((totGeneral+totCorporate)/TotRev)*100;
```

```
3001      f_ReportView.PrintText(cPage, 0.55, rTop+0.02, 8.0-(2*Std_ColSize), rBottom,
3002            '(G) Total Common Costs (D/A)', dt_Left+dt_WordBreak);
3003      f_ReportView.PrintText(cPage, 8.0-(2*Std_ColSize), rTop+0.02, 8.0, rBottom,
3004            Format('%5.2f',[TempNum])+'%', dt_Right+dt_WordBreak);
3005
3006      rTop:=rBottom+0.2;
3007      rBottom:=rTop+0.25;
3008      PageObj.FntStyle := [fsBold,fsUnderline];
3009      f_ReportView.PrintText(cPage, 0.55, rTop+0.02, 8.0-(2*Std_ColSize), rBottom,
3010            'Comparison Price Allocator (G / (100% - G))', dt_Left+dt_WordBreak);
3011
3012      rTop:=rBottom;
3013      rBottom:=rTop+0.25;
3014      PageObj.FntStyle := [fsBold];
3015      TempNum:=(totGeneral+totCorporate)/TotRev;
3016      TempNum:=(TempNum/(1-TempNum))*100;
3017      f_ReportView.PrintText(cPage, 0.55, rTop+0.02, 8.0-(2*Std_ColSize), rBottom,
3018            '(H)', dt_Left+dt_WordBreak);
3019      f_ReportView.PrintText(cPage, 0.85, rTop+0.02, 4.0, rBottom,
3020            '[% Total Common Costs]', dt_Center+dt_WordBreak);
3021      f_ReportView.PrintText(cPage, 8.0-(2*Std_ColSize), rTop+0.02, 8.0, rBottom,
3022            Format('%5.2f',[TempNum])+'%', dt_Right+dt_WordBreak);
3023      f_ReportView.PrintLine(cPage, 0.85, rBottom, 4.0, rBottom, 1);
3024      rTop:=rBottom;
3025      rBottom:=rTop+0.25;
3026      f_ReportView.PrintText(cPage, 0.85, rTop+0.02, 4.0, rBottom,
3027            '[100% - % Total Common Costs]', dt_Center+dt_WordBreak);
3028
3029      PageObj.FntStyle := [];
3030    end; {ComparisonDetail}
3031
3032  begin
3033    Std_ColSize := 10 / 12;
3034
3035    // print out the summary page
3036    Inc(Page);  // summary page
3037    sPage:=Page;
3038    f_ReportView.AddBookMark(0, Page, 'Common Cost Summary');
3039    MainHeader(sPage,'Summary of Common Costs');
```

```
3040      Footer(sPage);
3041      SummaryDetail;
3042
3043      // print out the Comparison Price Fixed Allocator
3044      Inc(Page);  // comparison price fixed allocator page
3045      cPage:=Page;
3046      f_ReportView.AddBookMark(0, Page, 'Comparison Price Fixed Allocator');
3047      MainHeader(cPage,'Comparison Price Fixed Allocator');
3048      Footer(cPage);
3049      ComparisonDetail;
3050    end; {SummaryReport}
3051
3052    // RFPC 0006
3053    // This function returns the description for the requested account
3054    function Tf_Expense.GetActDesc(Act:string):string;
3055    begin
3056      if (Act='611X') then Result:='Network Support Expenses'
3057      else if (Act='6110') then Result:='Network Support Expense'
3058      else if (Act='6112') then Result:='Motor Vehicle Expense'
3059      else if (Act='6113') then Result:='Aircraft Expense'
3060      else if (Act='6114') then Result:='Special Purpose Vehicles Expense'
3061      else if (Act='6115') then Result:='Garage Work Equipment Expense'
3062      else if (Act='6116') then Result:='Other Work Equipment Expense'
3063      else if (Act='612X') then Result:='General Support Expenses'
3064      else if (Act='6120') then Result:='General Support Expense'
3065      else if (Act='6121') then Result:='Land and Building Expense'
3066      else if (Act='6122') then Result:='Furniture and Artworks Expense'
3067      else if (Act='6123') then Result:='Office Equipment Expense'
3068      else if (Act='6124') then Result:='General Purpose Computers Expense'
3069      else if (Act='6112') then Result:='Motor Vehicle Expense'
3070      else if (Act='6113') then Result:='Aircraft Expense'
3071      else if (Act='6114') then Result:='Special Purpose Vehicle Expense'
3072      else if (Act='6115') then Result:='Garage Work Equipment Expense'
3073      else if (Act='6116') then Result:='Other Work Equipment Expense'
3074      else if (Act='6211') then Result:='Analog Electronic Expense'
3075      else if (Act='6212') then Result:='Digital Electronic Expense'
3076      else if (Act='6215') then Result:='Electromechanical Expense'
3077      else if (Act='6220') then Result:='Operator Systems Expense'
3078      else if (Act='6231') then Result:='Radio Systems Expense'
```

```
3079    else if (Act='6232') then Result:='Circuit Equipment Expense'
3080    else if (Act='6311') then Result:='Station Apparatus Expense'
3081    else if (Act='6341') then Result:='Large Private Branch Exchange Expense'
3082    else if (Act='6351') then Result:='Public Tel. Terminal Equip. Expense'
3083    else if (Act='6362') then Result:='Other Terminal Equipment Expense'
3084    else if (Act='6410') then Result:='Cable and Wire Facilities Expense'
3085    else if (Act='6411') then Result:='Poles Expense'
3086    else if (Act='6421') then Result:='Aerial Cable Expense'
3087    else if (Act='6422') then Result:='Underground Cable Expense'
3088    else if (Act='6423') then Result:='Buried Cable Expense'
3089    else if (Act='6424') then Result:='Submarine Cable Expense'
3090    else if (Act='6425') then Result:='Deep Sea Cable Expense'
3091    else if (Act='6426') then Result:='Intrabuilding Network Cable Expense'
3092    else if (Act='6431') then Result:='Aerial Wire Expense'
3093    else if (Act='6441') then Result:='Conduit Systems Expense'
3094    else if (Act='6510') then Result:='Other PP and E Expense'
3095    else if (Act='6511') then Result:='Prop. Held for Future Telecom Use Exp.'
3096    else if (Act='6512') then Result:='Provisioning Expense'
3097    else if (Act='653X') then Result:='Network Operations Expenses'
3098    else if (Act='6530') then Result:='Network Operations Expense'
3099    else if (Act='6531') then Result:='Power Expense'
3100    else if (Act='6532') then Result:='Network Administration Expense'
3101    else if (Act='6533') then Result:='Testing Expense'
3102    else if (Act='6534') then Result:='Plant Operations Admin. Expense'
3103    else if (Act='6535') then Result:='Engineering Expense'
3104    else if (Act='6540') then Result:='Access Expense'
3105    else if (Act='6560') then Result:='Depreciation and Amortization Expenses'
3106    else if (Act='6561') then Result:='Depr. Exp. - Telecom Plant in Service'
3107    else if (Act='6562') then Result:='Depr. Exp. - Prop. Held for Fut. Tel. Use'
3108    else if (Act='6563') then Result:='Amortization Expense - Tangible'
3109    else if (Act='6564') then Result:='Amortization Expense - Intangible'
3110    else if (Act='6565') then Result:='Amortization Expense - Other'
3111    else if (Act='66XX') then Result:='Customer Operations Marketing and Services Expenses'
3112    else if (Act='6610') then Result:='Marketing'
3113    else if (Act='6611') then Result:='Product Management'
3114    else if (Act='6612') then Result:='Sales'
3115    else if (Act='6613') then Result:='Product Advertising'
3116    else if (Act='6620') then Result:='Services'
3117    else if (Act='6621') then Result:='Call Completion Services'
```

```
3118     else if (Act='6622') then Result:='Number Services'
3119     else if (Act='6623') then Result:='Customer Services'
3120     else if (Act='671X') then Result:='Executive and Planning'
3121     else if (Act='6710') then Result:='Executive and Planning'
3122     else if (Act='6711') then Result:='Executive'
3123     else if (Act='6712') then Result:='Planning'
3124     else if (Act='672X') then Result:='General and Administrative'
3125     else if (Act='6720') then Result:='General and Administrative'
3126     else if (Act='6721') then Result:='Accounting and Finance'
3127     else if (Act='6722') then Result:='External Relations'
3128     else if (Act='6723') then Result:='Human Resources'
3129     else if (Act='6724') then Result:='Information Management'
3130     else if (Act='6725') then Result:='Legal'
3131     else if (Act='6726') then Result:='Procurement'
3132     else if (Act='6727') then Result:='Research and Development'
3133     else if (Act='6728') then Result:='Other General and Administrative'
3134     else if (Act='6790') then Result:='Provision for Uncollectible Notes Rec.'
3135     else if (Act='7100') then Result:='Other Operating Income and Expense'
3136     else if (Act='7110') then Result:='Income from Custom Work'
3137     else if (Act='7130') then Result:='Return from Nonreg. Use of Reg. Facil.'
3138     else if (Act='7140') then Result:='Gains and Losses from Foreign Exchange'
3139     else if (Act='7150') then Result:='G and L from Disposition of Land and Artworks'
3140     else if (Act='7160') then Result:='Other Operating Gains and Losses'
3141     else if (Act='7200') then Result:='Operating Taxes'
3142     else if (Act='7210') then Result:='Operating Investment Tax Credits - Net'
3143     else if (Act='7220') then Result:='Operating Federal Income Taxes'
3144     else if (Act='7230') then Result:='Operating State and Local Income Taxes'
3145     else if (Act='7240') then Result:='Operating Other Taxes'
3146     else if (Act='7250') then Result:='Provision for Def. Oper. Inc. Taxes - Net'
3147     else if (Act='671X') then Result:='Executive and Planning'
3148     else if (Act='672X') then Result:='Planning'
3149     else if (Act='611X') then Result:='Network Support Expenses'
3150     else if (Act='612X') then Result:='General Support Expenses'
3151     else if (Act='653X') then Result:='Network Operations Expenses'
3152     else if (Act='66XX') then Result:='Customer Operations Marketing and Services Expenses'
3153     else if (Act='2110') then Result:='Land and Support Assets'
3154     else if (Act='2111') then Result:='Land'
3155     else if (Act='2112') then Result:='Motor Vehicles'
3156     else if (Act='2113') then Result:='Aircraft'
```

```
3157      else if (Act='2114') then Result:='Special Purpose Vehicles'
3158      else if (Act='2115') then Result:='Garage Work Equipment'
3159      else if (Act='2116') then Result:='Other Work Equipment'
3160      else if (Act='2121') then Result:='Buildings'
3161      else if (Act='2122') then Result:='Furniture'
3162      else if (Act='2123') then Result:='Office Equipment'
3163      else if (Act='2124') then Result:='General Purpose Computers'
3164      else Result:='';
3165    end; {GetActDesc}
3166
3167    end.
3168
```

INTEROFC.pas

```
2
3   unit Interofc;
4   {*************************************************************************
5     This unit contains the routines to handle all of the Interoffice investment
6     calculations.
7   *************************************************************************}
8
9   { RFPC 0024  -  Reformulation of Multiplexing Equipment VIS Investment
10    Originated 01/20/98
11
12    1. The change will reformulate the VIS investment calculation for the 31
13       multiplexers (IHMUX31 and IRMUX31 cost components) and 31 digital
14       cross-connect systems (IHDCS31 and IRDCS31 cost components).
15    2. The change will replace the IHMUX31DS3, IRMUX31DS3, IHDCS31DS3, and
16       IRDCS31DS3 components with new IHMUX31DS1, IRMUX31DS1, IHDCS31DS1, and
17       IRDCS31DS1 components.
18
19    Description of the program change desired:
20
21    1. Change all 'DSX3' references to 'DSX1',
22    2. Change 'DS3' reference to 'DS1',
23    3. Change all 'DSX3' references to 'DSX1' in the following sections:
24
25    DCS equipment and multiplexers.
26
27   ------------------------------------------------------------------------
28    RFPC 0031  -  Addition of Interoffice Transport Demand Components
29    Originated 2/25/98
30
31    1. The change will allow better utilization and allocation of investments
32       between the various digital signal levels.
33
34    Description of the program change desired:
35
36    1. Add IHDS0, IRDS0, IHDS1, IRDS1, IHDS3, IRDS3, IHSWPORTS, IRSWPORTS
```

```
37       interoffice transport demand unit components.
38
39    ------------------------------------------------------------------------
40    RFPC0032: Revised Interoffice Total Investments and Unit Values
41
42      Modify volume-sensitive component cost equations for transport
43      circuit equipment so that methodology is more consistent with
44      capacity cost principles.  That is, use equations of the general
45      form (Investment per Common Equipment / Common Equipment DS-0
46      Capacity) + (Inv per Plug-in / Plug-in DS-0 Capacity) for volume-
47      sensitive costs per unit.
48
49      Adjust the transport equipment sizing and average unit calculations
50      to account for the SS7 A links that are assumed to terminate at
51      each host/standalone office.
52
53      Revise mapping unitization of circuit equipment, fiber cable,
54      and structure investments so that all units (average, volume-
55      sensitive, and volume-insensitive) reflect total demand for
56      DS0-equivalent channels, and unit investments for DS1s and
57      DS3s are then converted appropriately using 24 and 672 channels
58      per unit, respectively .
59
60    * How will the desired change enhance the Integrated Cost Model (ICM)
61
62      Volume-sensitive investments will be more consistent with capacity
63      cost principles.
64
65      For both unit and total investments at each office and at a state
66      level, the sum of volume-sensitive and volume-insensitive investments
67      will always equal average investments.
68
69      Mapping processes for apportioning shared equipment to DS-0, DS-1,
70      and DS-3 terminations will be simplified while still consistent
71      with provisioning and usage of the equipment.
72
73      Average SS7 Link costs will be more accurate and will not vary by
74      office as significantly as in the past.
75
```

```
76    ------------------------------------------------------------------------
77    RFPC0054: Modification of DCS 3/1 Common Equipment Sizing
78
79    Increase accuracy of Interoffice termination investment calculations.
80
81    ------------------------------------------------------------------------
82    RFPC0052a: Modification of Investment Calculations at Sites with Collocated
83               Class 4 and Class 5 Switches.
84
85    Allow Interoffice Module to recognize and use the fiber outside plant
86    associated with Class 5 switch at locations with colocated Class 4 and Class 5
87    switches. In addition, all circuit equipment of tandem will be associated with
88    the Class 5 CLLI.
89
90    ------------------------------------------------------------------------
91    RFPC0059: Modification of IHRING Component Calculation.
92
93    Change the calculation for the IHRING component to take tandem into account.
94
95    ------------------------------------------------------------------------
96    RFPC0067: Addition of InterRing components.
97
98    Generate additional circuit equipment and facilities components to interconnect
99    individual rings to each other.
100
101   }
102   interface
103
104   uses WinTypes, WinProcs, Classes, Graphics, Forms, Controls, StdCtrls, Tabs,
105     Buttons, ExtCtrls, Mask, DBCtrls, DB, DBTables, Menus, Grids, DBGrids,
106     Dialogs, SysUtils, Gauges, Printers, Runtopt, Query, Dbialias, Expense,
107     RepView, GenPrnt, Import, Gen_Exp, Invntry, Monitr, Math, OSPTypes,
108     OSPInvest, ICMDM;
109
110   type
111     TIOFInvestment = Object
112             t_Dist    : TTable;
113
114             constructor Init;
```

```
115             destructor  Done;
116             procedure   Make;
117             procedure   Open;
118             procedure   Write (CLLI, Account, Component, Category : String; Inv, Vs, Vis : Double;
119                                writeRemotes: boolean);
120       end;
121
122     Tf_InterOfc = class(TForm)
123       dsHst2Rem: TDataSource;
124       dsHst2Hst: TDataSource;      // Created database table - host to host (acth2h.db)
125       pnlEntry: TPanel;           // screen displaying database table and tab selections
126       pnlControl: TPanel;         // screen portion with navigator buttons, sort order button, update
127   association, & process button
128       IOFMenu: TMainMenu;
129       File1: TMenuItem;
130       Build: TMenuItem;
131       Open: TMenuItem;
132       N1: TMenuItem;
133       Import1: TMenuItem;
134       Export1: TMenuItem;
135       N2: TMenuItem;
136       Exit1: TMenuItem;
137       Help1: TMenuItem;
138       l_CopyRight: TLabel;
139       od_Iof: TOpenDialog;
140       DBNavigator1: TDBNavigator;
141       cb_Assoc: TCheckBox;
142       sd_Iof: TSaveDialog;
143       btnProcess: TButton;
144       cbSort: TComboBox;
145       Label5: TLabel;      // Created database table - host to remote (acth2r.db)
146       N3: TMenuItem;
147       Print1: TMenuItem;
148       t_Grid: TTable;             // Run time options database for outside plant run (roosp.db)
149       pnlReport: TPanel;          // screen portion which shows report progress
150       GroupBox2: TGroupBox;
151       RptGauge: TGauge;
152       btnCancel: TBitBtn;          // Host to Remote database table (3008)
153       t_IOFSet: TTable;           // User option database table (iofset.db)
```

```
154       dsIOFSet: TDataSource;
155       TabSet: TTabSet;
156       DBGrid1: TDBGrid;
157       pnlIndividual: TPanel;      // screen which has selected equip buttons, user parameter inputs, & report
158   button
159       GroupBox3: TGroupBox;
160       Label1: TLabel;
161       edUti1: TEdit;
162       btnReport: TBitBtn;
163       Query1: TMenuItem;
164       Label2: TLabel;
165       edCblLen: TEdit;
166       Label3: TLabel;
167       edCblCost: TEdit;
168       rgEquipment: TRadioGroup;
169       pnlBuild: TPanel;           // screen portion where you can input new table names & Input text files
170       Label4: TLabel;
171       Panel2: TPanel;             // cosmetic blank bar on plnBuild panel
172       GroupBox1: TGroupBox;
173       Label6: TLabel;
174       Label7: TLabel;
175       edH2HDB: TEdit;
176       edH2RDB: TEdit;
177       GroupBox4: TGroupBox;
178       Label8: TLabel;
179       Label9: TLabel;
180       edUSDEMD: TEdit;
181       edH2HCSV: TEdit;
182       btnBldCancel: TBitBtn;
183       btnBldOK: TBitBtn;
184       sbH2H: TSpeedButton;
185       sbH2R: TSpeedButton;
186       sbUswD: TSpeedButton;
187       sbH2HDef: TSpeedButton;
188       PrintDialog1: TPrintDialog;
189       Label10: TLabel;
190       edTndCSV: TEdit;
191       sbTndDef: TSpeedButton;
192       Common1: TMenuItem;
```

```
193       Tab1: TMenuItem;
194       dsTandems: TDataSource;
195       Label11: TLabel;
196       edTndDB: TEdit;
197       sbTnd: TSpeedButton;
198       dsInterRing: TDataSource;
199       Label13: TLabel;
200       edIRDB: TEdit;
201       sbIR: TSpeedButton;
202       Label14: TLabel;
203       edIRCSV: TEdit;
204       sbIRCSV: TSpeedButton;          // Switches database table (2001)
205       procedure FormCreate(Sender: TObject);
206       procedure TabSetClick(Sender: TObject);
207       procedure Exit1Click(Sender: TObject);
208       procedure Help1Click(Sender: TObject);
209       procedure FormShow(Sender: TObject);
210       procedure OpenClick(Sender: TObject);
211       procedure Export1Click(Sender: TObject);
212       procedure BuildTables;
213       procedure btnProcessClick(Sender: TObject);
214       procedure cbSortChange(Sender: TObject);
215       procedure btnCancelClick(Sender: TObject);
216       procedure Print1Click(Sender: TObject);
217       procedure FormResize(Sender: TObject);
218       procedure Query1Click(Sender: TObject);
219       procedure FormActivate(Sender: TObject);
220       procedure F1Help(Sender: TObject; var Key: Word; Shift: TShiftState);
221       procedure btnBldCancelClick(Sender: TObject);
222       procedure btnBldOKClick(Sender: TObject);
223       procedure DoesFileExist(Sender: TObject);
224       procedure BuildClick(Sender: TObject);
225       procedure sbH2HClick(Sender: TObject);
226       procedure sbH2RClick(Sender: TObject);
227       procedure sbUswDClick(Sender: TObject);
228       procedure sbH2HDefClick(Sender: TObject);
229       procedure t_Hst2RemBeforePost(DataSet: TDataSet);
230       procedure t_Hst2HstBeforePost(DataSet: TDataSet);
231       procedure sbTndDefClick(Sender: TObject);
```

```
232       procedure Common1Click(Sender: TObject);
233       procedure Tab1Click(Sender: TObject);
234       procedure sbTndClick(Sender: TObject);
235       procedure sbIRClick(Sender: TObject);
236       procedure sbIRCSVClick(Sender: TObject);
237     private
238       procedure CalcCl1iInvest(t_IOF,t_tandems:tTable; ClliCode:String;
239                                           FacType: Char;CRing: Integer);
240       procedure BuildRem2Hst;
241       procedure CalcADMOCComms(t_tandems: TTable);
242       procedure CalcInterRingADMInvestments;    // RFPC0067
243       procedure DeleteTempTables;
244       function  IsColocated(Clli: string; t_tandems: TTable):boolean;
245     public
246        procedure SetGridConstants;
247        procedure Entry;
248        procedure Process;
249        procedure PrintIOF( Open : Boolean);
250        function  CalcDistance(CLat, CLong, HLat, HLong:Real): Integer;
251        function  ArcCos(x: double): double;
252        function  SelectFile(Title,DefaultExt,Filter,InitialDir: String): String;
253        procedure ProcessRingDistance;
254     end;
255
256   var
257     f_InterOfc: Tf_InterOfc;
258     H2HTabName,H2RTabName,          // H2HTabName: Host to Host filename (3007)
259     TndTabName,IRTabName : String;  // H2RTabName: Host to Remote filename (3008)
260                                     // TndTabName: Tandems filename (3011)
261                                     // IRTabName:  InterRing filename (3012)
262     CancelProcess, InForm: Boolean;    // CancelProcess: cancel button selected
263                                        // InForm:
264     iADMIN_FILL: Double;     // runtime set (iofset.db), adjusts the number of interoffice
265                              //  circuits to account for maintenance, spare & defective material
266     IntraRingFactor: Double; // runtime set (iofset.db), percent of traffic that originates
267                              //  and terminates on the same ring
268     HelpSubject: Longint;    // =13
269     Facility_Length : double;  //
270     tFile: TextFile;         // log file for ring distance calculation (hostdist.log)
```

```
271     t_IArcFile: TTable;      // interoffice arc file
272
273     IOF_Table : TIOFInvestment;  // investment table
274   implementation
275   uses Assoc, GlobalDa, Nodes,  Ospmater, OSPLabor, nemain,
276        Gen_Impt, IOFUtils, Demand;
277
278   {$R *.DFM}
279
280   var ADMOC48Comms, ADMOC12Comms, IIADMOC48Comms,
281       IIADMOC12Comms, IISWPorts : array [1 .. 100] of SmallInt;
282       HubCllis, iHubCllis, iHubCllisSub, iCllis, iCllisSub, iCllisHub : TStringList;
283
284    constructor TIOFInvestment.Init;
285     begin
286       t_Dist := TTable.Create(Nil);
287     end;
288
289    destructor TIOFInvestment.Done;
290     begin
291       if t_Dist.Active then t_Dist.Close;
292       t_Dist.Destroy;
293     end;
294
295    procedure TIOFInvestment.Write(CLLI, Account, Component, Category : String; Inv, Vs, Vis : Double;
296                                   writeRemotes: boolean);
297      procedure WriteRec(Clli:string);
298        begin
299          if t_Dist.FindKey([CLLI, Account, Component])
300            then begin
301                   t_Dist.Edit;
302                   t_Dist.FieldByName('Total Inv').AsFloat  := t_Dist.FieldByName('Total Inv').AsFloat + Inv;
303                   t_Dist.FieldByName('VS Inv').AsFloat     := t_Dist.FieldByName('VS Inv').AsFloat    + Vs;
304                   t_Dist.FieldByName('VIS Inv').AsFloat    := t_Dist.FieldByName('VIS Inv').AsFloat   + Vis;
305                   t_Dist.Post;
306                 end
307            else begin
308                   t_Dist.Insert;
309                   t_Dist.FieldByName('CLLI').AsString      := Clli;
```

```
310                 t_Dist.FieldByName('Account').AsString   := Account;
311                 t_Dist.FieldByName('Component').AsString := Component;
312                 t_Dist.FieldByName('Category').AsString  := Category;
313                 t_Dist.FieldByName('Total Inv').AsFloat  := Inv;
314                 t_Dist.FieldByName('VS Inv').AsFloat     := Vs;
315                 t_Dist.FieldByName('VIS Inv').AsFloat    := Vis;
316                 t_Dist.Post;
317               end;
318     end; {WriteRec}
319   begin
320    t_Dist.IndexName := '';
321    // write out the record
322    WriteRec(Clli);
323
324    if not writeRemotes then exit;
325
326    // if the Clli has any remotes then write out duplicate records for those remotes
327    with dm_ICM.t_Hst2Rem do
328      begin
329        if not Active then Open;
330        IndexName:='';
331        SetRange([Clli],[Clli]);
332        First;
333        if (RecordCount > 0) then
334          while not eof do
335            begin
336              WriteRec(FieldByName('RClli').AsString);
337              Next;
338            end;
339        CancelRange;
340      end;
341   end;
342
343  procedure TIOFInvestment.Make;
344   begin
345  { Create a table to hold IOF investment on per line basis }
346
347   with t_Dist do
348     begin
```

```
349         Active := False;
350         DatabaseName := 'modata';
351         TableName := 'actH2H';
352         TableType := ttParadox;
353         Exclusive := True;
354
355         with FieldDefs do
356           begin
357             Clear;
358             Add ('CLLI',          FTString, 11, True);
359             Add ('Account',       FTString, 6,  True);
360             Add ('Component',     FTString, 16, False);
361             Add ('Category',      FTString, 1, False);
362             Add ('Total Inv',     FTFloat,  0,  False);
363             Add ('VS Inv',        FTFloat,  0,  False);
364             Add ('VIS Inv',       FTFloat,  0,  False);
365           end;
366
367         with IndexDefs do
368           begin
369             Clear;
370             Add('Main', 'CLLI;Account;Component', [ixPrimary, ixUnique]);
371             Add('ByAcc', 'Account;Component', [ixCaseInsensitive]);
372             Add('ByCat', 'Category;Account;Component', [ixCaseInsensitive]);
373           end;
374
375         CreateTable;
376         Open;
377       end;
378   end;
379
380   procedure TIOFInvestment.Open;
381    begin
382  { Open table to hold interoffice investments }
383
384    with t_Dist do
385      begin
386        Active := False;
387        DatabaseName := 'modata';
```

```
388         TableName := 'actH2H';
389         TableType := ttParadox;
390         Exclusive := True;
391         Open;
392       end;
393   end;
394
395   {--------------------------------------------------------------------------
396   Form create event
397   --------------------------------------------------------------------------}
398   procedure Tf_InterOfc.FormCreate(Sender: TObject);
399   begin
400     {set default datasource to default page}
401     dbNavigator1.DataSource := dsHst2Hst;
402     InForm := False;
403     f_InterOfc.Height := 451;  // set form height
404     HelpSubject := 13;
405   end; {procedure Tf_InterOfc.FormCreate}
406
407   {--------------------------------------------------------------------------
408   Activate form event
409   --------------------------------------------------------------------------}
410   procedure Tf_InterOfc.FormActivate(Sender: TObject);
411   var Alias : TDBIAlias;
412   begin
413     {set default file/table open and save path to Alias path}
414     Alias := tDbiAlias.Create;
415     AliasPath := Alias.GetPath('modata');
416     Alias.Free;
417     UnitName := 'InterOfc';
418   end;
419
420   {--------------------------------------------------------------------------
421   Form show event
422   Associate tables and open.
423   --------------------------------------------------------------------------}
424   procedure Tf_InterOfc.FormShow(Sender: TObject);
425   begin
426     {get host to host database filename & location}
```

```
427     H2HTabName := f_associate.GetFileName(3007);
428     {get host to remote database filename & location}
429     H2RTabName := f_associate.GetFileName(3008);
430     {get tandems database filename & location}
431     TndTabName := f_associate.GetFileName(3011);
432     {get InterRing database filename & location}
433     IRTabName  := f_associate.GetFileName(3012);
434     {get ICM copyright information}
435     l_Copyright.Caption := f_associate.GetFileName(5);
436     {set form title using host to host database filename}
437     Caption := 'Inter-Office Transport ['+ ExtractFilename(H2HTabName)+']';
438     RunFlag := False;  // don't run interoffice process
439     cb_Assoc.Checked := False;    // clear the update association check box
440     pnlReport.Visible := False;  // don't show the percent progress panel
441     pnlEntry.Visible := True;     // display the database table screen
442     pnlControl.Visible := True;  // display the panel with the navigator buttons
443     TabSet.Visible := True;       // display the tab selections
444     TabSet.TabIndex := 0;
445     Tab1.Caption := TabSet.Tabs[0];
446     {open the host to host database table}
447     with dm_ICM.t_Hst2Hst do begin
448       Active := False;
449       DatabaseName := ExtractFilePath(H2HTabName);  // get the file location
450       TableName := ExtractFilename(H2HTabName);     // get the table filename
451       TableType := ttDefault;
452       Exclusive := True;
453       Open;
454     end; // end [with t_hst2hst]
455     {open the host to remote database table}
456     with dm_ICM.t_Hst2Rem do begin
457       Active := False;
458       DatabaseName := ExtractFilePath(H2RTabName);  // get the file location
459       TableName := ExtractFilename(H2RTabName);     // get the table filename
460       TableType := ttDefault;
461       Exclusive := True;
462       Open;
463     end;  // end [with t_hst2rem]
464     {open the tandems database table}
465     with dm_ICM.t_Tandems do begin
```

```
466       Active := False;
467       DatabaseName := ExtractFilePath(TndTabName);  // get the file location
468       TableName := ExtractFilename(TndTabName);     // get the table filename
469       TableType := ttDefault;
470       Exclusive := True;
471       Open;
472     end;  // end [with t_tandems]
473     {open the InterRing database table}
474     with dm_ICM.t_InterRing do begin
475       Active := False;
476       DatabaseName := ExtractFilePath(IRTabName);  // get the file location
477       TableName := ExtractFilename(IRTabName);     // get the table filename
478       TableType := ttDefault;
479       Exclusive := True;
480       Open;
481     end;  // end [with t_InterRing]
482   end; {procedure Tf_InterOfc.FormShow}
483
484   {-------------------------------------------------------------------------
485   Form entry point: display the form
486   -------------------------------------------------------------------------}
487   procedure Tf_InterOfc.Entry;
488   begin
489     Mode := 1;  // not processing mode
490     f_InterOfc.Height := 451;  // set form height
491     Show;       // display the form
492   end; {procedure Tf_InterOfc.Entry}
493
494   {-------------------------------------------------------------------------
495   Display Help on F1 keypress
496   -------------------------------------------------------------------------}
497   procedure Tf_InterOfc.F1Help(Sender: TObject; var Key: Word;
498     Shift: TShiftState);
499   begin
500     case key of
501       vk_F1:  Application.HelpCommand(HELP_CONTEXT, HelpSubject);
502      end;
503   end;  {procedure Tf_InterOfc.F1Help}
504
```

```
505  {-------------------------------------------------------------------------
506  Respond to a tab click by switching active tables
507  -------------------------------------------------------------------------}
508  procedure Tf_InterOfc.TabSetClick(Sender: TObject);
509  begin
510    if TabSet.TabIndex = 0 then begin  // is host to host tab selected (index=0)
511      pnlIndividual.Visible := False;  // don't display selected equip/user params screen
512      dbGrid1.Visible := True;         // display database table screen
513      dbNavigator1.DataSource := dsHst2Hst;  // set up information for host to host data
514      dbGrid1.DataSource := dsHst2Hst; // set up information for host to host data
515      H2HTabName := f_associate.GetFileName(3007);  // get host to host filename from accociate (3007)
516      {set form title using host to host filename}
517      Caption := 'Inter-Office Transport ['+ ExtractFilename(H2HTabName)+']';
518      cbSort.Items.Clear;  // clear data items in sort field box
519      cbSort.Items.Add('By End Clli Code');      // add new item to sort field box
520      cbSort.Items.Add('By Tandem Clli Code');  // add new item to sort field box
521      cbSort.Text := 'By End Clli Code';        // indicate default sort item in box
522    end  // end [if TabSet.TabIndex=0]
523    else if TabSet.TabIndex = 1 then begin  // is host to remote tab selected (index=1)
524      pnlIndividual.Visible := False;  //  don't display selected equip/user params screen
525      dbGrid1.Visible := True;         // display database table screen
526      dbNavigator1.DataSource := dsHst2Rem;  // set up information for host to remote data
527      dbGrid1.DataSource := dsHst2Rem; // set up information for host to remote data
528      H2RTabName := f_associate.GetFileName(3008);  // get host to remote filename from accociate (3008)
529      {set form title using host to remote filename}
530      Caption := 'Inter-Office Transport ['+ ExtractFilename(H2RTabName)+']';
531      cbSort.Items.Clear;  // clear data items in sort field box
532      cbSort.Items.Add('By Host/Remote');       // add new item to sort field box
533      cbSort.Items.Add('By Host Clli Code');    // add new item to sort field box
534      cbSort.Items.Add('By Remote Clli Code');  // add new item to sort field box
535      cbSort.Text := 'By Host/Remote';          // indicate default sort item in box
536    end  // end [if TabSet.TabIndex=1]
537    else if TabSet.TabIndex = 2 then begin  // is tandems tab selected (index=2)  //RFPC0052A
538      pnlIndividual.Visible := False;  //  don't display selected equip/user params screen
539      dbGrid1.Visible := True;         // display database table screen
540      dbNavigator1.DataSource := dsTandems;  // set up information for tandems data
541      dbGrid1.DataSource := dsTandems; // set up information for tandems data
542      TndTabName := f_associate.GetFileName(3011);  // get tandems filename from accociate (3011)
543      {set form title using tandem to tandem filename}
```

```
544       Caption := 'Inter-Office Transport ['+ ExtractFilename(TndTabName)+']';
545       cbSort.Items.Clear;  // clear data items in sort field box
546       cbSort.Text := '';   // blank out the sort field box entry
547     end  // end [if TabSet.TabIndex=2]
548     else if TabSet.TabIndex = 3 then begin  // is InterRing tab selected (index=3)  //RFPC0052A
549       pnlIndividual.Visible := False;  //  don't display selected equip/user params screen
550       dbGrid1.Visible := True;         // display database table screen
551       dbNavigator1.DataSource := dsInterRing;  // set up information for InterRing data
552       dbGrid1.DataSource := dsInterRing; // set up information for InterRing data
553       IRTabName := f_associate.GetFileName(3012);  // get InterRing filename from accociate (3011)
554       {set form title using tandem to tandem filename}
555       Caption := 'Inter-Office Transport ['+ ExtractFilename(IRTabName)+']';
556       cbSort.Items.Clear;  // clear data items in sort field box
557       cbSort.Text := '';   // blank out the sort field box entry
558     end  // end [if TabSet.TabIndex=3]
559     else if TabSet.TabIndex = 4 then begin
560       dbGrid1.Visible := False;
561       pnlIndividual.Visible := True;
562     end;  // end [if TabSet.TabIndex=4]
563
564   Tab1.Caption := TabSet.Tabs[TabSet.TabIndex]; // set help menu item
565 end; {procedure Tf_InterOfc.TabSetClick}
566
567 {-------------------------------------------------------------------------
568 Exit the form, cleanup and close tables
569 -------------------------------------------------------------------------}
570 procedure Tf_InterOfc.Exit1Click(Sender: TObject);
571 begin
572   if cb_Assoc.Checked then begin    { reset both tables }
573     {place host to host database filename with associated index (3007) in usetable database}
574     f_associate.SetFileName(3007, H2HTabName);
575     {place host to remote database filename with associated index (3008) in usetable database}
576     f_associate.SetFileName(3008, H2RTabName);
577     {place tandems database filename with associated index (3011) in usetable database}
578     f_associate.SetFileName(3011, TndTabName);
579     {place InterRing database filename with associated index (3012) in usetable database}
580     f_associate.SetFileName(3012, IRTabName);
581   end;  // end [if cb_Assoc.Checked]
582   cb_Assoc.Checked := False;  // clear update association box
```

```
583
584     if RunFlag then  // is run flag set
585       f_associate.SetFileName(100, 'YES'); // yes, set process demand to yes in usetable database
586     Hide;  // do not display the form
587     dm_ICM.t_Hst2Hst.Close;      // close host to host database table
588     dm_ICM.t_Hst2Rem.Close;      // close host to remote database table
589     dm_ICM.t_Tandems.Close;      // close tandems database table
590     dm_ICM.t_InterRing.Close;    // close InterRing database table
591   end; {procedure Tf_InterOfc.Exit1Click}
592
593   {-------------------------------------------------------------------------
594   export active notebook table to text file
595   -------------------------------------------------------------------------}
596   procedure Tf_InterOfc.Export1Click(Sender: TObject);
597   begin
598     if TabSet.TabIndex = 0 then begin  // is host to host tab selected (index=0)
599       ExportTable(dm_ICM.t_Hst2Hst);    // export host to host table to text file
600       WeAreWatching.LogEvent(mNORMAL,'Export1Click',' Host to Host Table ['+dm_ICM.t_Hst2Hst.TableName+']
601   Exported.')
602     end  // end [if TabSet.TabIndex=0]
603     else if TabSet.TabIndex = 1 then begin  // else host to remote tab selected (index=1)
604       ExportTable(dm_ICM.t_Hst2Rem);    // export host to remote table to text file
605       WeAreWatching.LogEvent(mNORMAL,'Export1Click',' Host to Remote Table ['+dm_ICM.t_Hst2Rem.TableName+']
606   Exported.')
607     end  // end else (if TabSet.TabIndex=1)
608     else if TabSet.TabIndex = 2 then begin  // else tandems tab selected (index=2)
609       ExportTable(dm_ICM.t_Tandems);    // export tandems table to text file
610       WeAreWatching.LogEvent(mNORMAL,'Export1Click',' Tandems Table ['+dm_ICM.t_Tandems.TableName+']
611   Exported.')
612     end  // end else (if TabSet.TabIndex=2)
613     else if TabSet.TabIndex = 3 then begin  // else InterRing tab selected (index=3)
614       ExportTable(dm_ICM.t_InterRing);    // export InterRing table to text file
615       WeAreWatching.LogEvent(mNORMAL,'Export1Click',' InterRing Table ['+dm_ICM.t_InterRing.TableName+']
616   Exported.')
617     end;  // end else (if TabSet.TabIndex=3)
618   end; {procedure Tf_InterOfc.Export1Click}
619
620   {-------------------------------------------------------------------------
621   Delete all temporary tables created during the build   RFPC0052a
```

```
622  ------------------------------------------------------------------------------}
623  procedure Tf_InterOfc.DeleteTempTables;
624  var TempTable: TTable;
625  begin
626    TempTable:=TTable.Create(nil);
627    with TempTable do
628      begin
629        Active := False;
630        DatabaseName := 'modata';
631        TableType := ttParadox;
632        // delete the temp unswitched demand table
633        TableName := 'IOTDEMND';
634        if Exists then DeleteTable;
635
636        Active := False;
637        // delete the temp h2h def table
638        TableName := 'H2H-DEF';
639        if Exists then DeleteTable;
640      end;
641    TempTable.Free;
642  end;
643
644  {------------------------------------------------------------------------------
645  Create a new table based on the active notebook table
646  ------------------------------------------------------------------------------}
647  procedure Tf_InterOfc.BuildTables;
648  begin
649    // build the remotes
650    if edH2RDB.Text = '' then begin  // is the host to remote text input field empty
651      MessageDlg('You must specify a Host to Remote filename.',mtError,[mbOK],0);  // display error
652      exit;  // exit procedure buildtables
653    end;  // end [if edh2rdb.text='']
654    cb_Assoc.Checked := True;  // set update association box
655    {set up the new host to remote table}
656    with dm_ICM.t_Hst2Rem do begin
657      Close;
658      Active := False;
659      DatabaseName := 'modata';
660      TableName := edH2RDB.Text;  // get filename from host to remote input field
```

```
661       TableType := ttParadox;
662       Exclusive := True;
663       with FieldDefs do begin     // set the new table fields
664         Clear;
665         Add ('HCLLI',   ftString, 11, True);
666         Add ('RCLLI',   ftString, 11, False);
667         Add ('SWPorts', ftInteger, 0, False);
668         Add ('DS0SAL',  ftInteger, 0, False);
669         Add ('DS1SAL',  ftInteger, 0, False);
670         Add ('DS3SAL',  ftInteger, 0, False);
671       end;  // end [with fielddefs]
672       with IndexDefs do begin  // set the new table index (sort) values
673         Clear;  // clear any existing indexes
674         Add('Main', 'HCLLI;RCLLI',[ixPrimary, ixUnique]);
675         Add('RCLLI', 'RCLLI', [ixCaseInsensitive]);
676         Add('HCLLI', 'HCLLI', [ixCaseInsensitive]);
677       end;  // end [with indexdefs]
678       CreateTable;  // create the new table
679       Open;  // open the new table
680       cb_Assoc.Checked := True;  // set the update association box
681     end;  // end [with t_hst2rem]
682
683     // create the arc table
684     t_IArcFile := TTable.Create(nil); {will contain info passed to the clustering routine}
685     CreateArcFile(t_IArcFile);
686     if not t_IArcFile.Active then t_IArcFile.Open;
687
688     BuildRem2Hst;  // build remote to host table from nodes table
689     {place new host to remote database filename with associated index (3008) in usetable database}
690     f_associate.SetFilename(3008,edH2RDB.Text);
691
692     if edTndDB.Text = '' then begin  // is tandems input text field empty
693       MessageDlg('You must specify a Tandems filename.',mtError,[mbOK],0);  // display error
694       exit;  // exit procedure buildtables
695     end;  // end [if edtnddb.text='']
696
697     with dm_ICM.t_Tandems do begin
698       if Active then Close;
699       DatabaseName := 'modata';
```

```
700       TableName := edTndDB.Text;
701       TableType := ttParadox;
702       Exclusive := True;
703       with FieldDefs do begin
704         Clear;
705         Add ('TCLLI',       ftString,  11, False);
706         Add ('ColoClli',    ftString,  11, False);  // RFPC0052a
707         Add ('TndCompany', ftString,   1, False);  // RFPC0052a
708         Add ('Lat',         ftFloat,    0, False);
709         Add ('Long',        ftFloat,    0, False);
710       end;
711       with IndexDefs do begin
712         Clear;
713         Add('Main', 'TCLLI', [ixPrimary,ixUnique]);
714         Add('byColoClli', 'ColoClli', [ixCaseInsensitive]);
715       end;
716       CreateTable;
717       Open;
718       cb_Assoc.Checked := True;  // set the update association box
719     end;
720     {place new tandem to tandem database filename with associated index (3011) in usetable database}
721     f_associate.SetFilename(3011,edTndDB.Text);
722
723     if edIRDB.Text = '' then begin  // is InterRing input text field empty
724       MessageDlg('You must specify an InterRing filename.',mtError,[mbOK],0);  // display error
725       exit;  // exit procedure buildtables
726     end;  // end [if edirdb.text='']
727
728     with dm_ICM.t_InterRing do begin
729       if Active then Close;
730       DatabaseName := 'modata';
731       TableName := edIRDB.Text;
732       TableType := ttParadox;
733       Exclusive := True;
734       with FieldDefs do begin
735         Clear;
736         Add ('ICLLI',                    ftString,  11, True);
737         Add ('ICLLI Substitution',       ftString,  11, False);
738         Add ('IHubCLLI',                 ftString,  11, False);
```

```
739         Add ('IHubCLLI Substitution',   ftString,  11, False);
740         Add ('IRing',                   ftInteger,  0, False);
741         Add ('ISWPorts',                ftInteger,  0, False);
742       end;
743       with IndexDefs do begin
744         Clear;
745         Add('Main', 'ICLLI', [ixPrimary,ixUnique]);
746         Add('byHubClli', 'IHubCLLI;iRing', [ixCaseInsensitive]);
747         Add('bySubstitution', 'ICLLI Substitution', [ixCaseInsensitive]);
748       end;
749       CreateTable;
750       Open;
751       cb_Assoc.Checked := True;  // set the update association box
752     end;
753     {place new tandem to tandem database filename with associated index (3011) in usetable database}
754     f_associate.SetFilename(3012,edIRDB.Text);
755
756     if edH2HDB.Text = '' then begin  // is host to host input text field empty
757       MessageDlg('You must specify a Host to Host filename.',mtError,[mbOK],0);  // display error
758       exit;  // exit procedure buildtables
759     end;  // end [if edh2hdb.text='']
760
761     {set up the new host to host table}
762     with dm_ICM.t_Hst2Hst do begin
763       Active := False;
764       DatabaseName := 'modata';
765       TableName := edH2HDB.Text;  // get filename from host to host input field
766       TableType := ttParadox;
767       Exclusive := True;
768       with FieldDefs do begin     // set new table fields
769         Clear;
770         Add ('ECLLI',      ftString,  11, True);
771         Add ('HubCLLI',    ftString,  11, False);     // RFPC0052a
772         Add ('GCLLI',      ftString,  11, False);
773         Add ('Ring',       ftInteger,  0, False);
774         Add ('RemoteSW',   ftInteger,  0, False);
775         Add ('SWPorts',    ftInteger,  0, False);
776         Add ('DS0SAL',     ftInteger,  0, False);
777         Add ('DS1SAL',     ftInteger,  0, False);
```

```
778         Add ('DS3SAL',     ftInteger,  0, False);
779       end;  // end {with fielddefs}
780       with IndexDefs do begin   // set table index (sort) values
781         Clear;  // clear any existing indexes
782         Add('ECLLI',   'ECLLI', [ixPrimary,ixUnique]);
783         Add('ByHubCLLI', 'HubCLLI;Ring', [ixCaseInsensitive]);
784       end;  // end {with indexdefs}
785       CreateTable;  // create the new table
786       Open;  // open the new table
787       BuildHst2Hst(dm_ICM.t_Hst2Hst,t_IArcFile);   // build host to host table from the wire center table
788
789       // close and free up the t_ArcFile table
790       UpdateTCllis(dm_ICM.t_Hst2Hst,t_IArcFile);   // first update the TCllis containing 'H2R'
791       t_IArcFile.Close;
792       t_IArcFile.Free;
793
794       {place new host to host database filename with associated index (3007) in usetable database}
795       f_associate.SetFilename(3007,edH2HDB.Text);
796       cb_Assoc.Checked := True;  // set the update association box
797     end;  // end [with t_hst2hst]
798
799     {Cleanup all temporary tables  RFPC0052a}
800     DeleteTempTables;
801
802     {reset the calculation flag for the demand}
803     f_associate.SetFileName(100, 'YES');
804   end;   {procedure Tf_InterOfc.NewClick}
805
806   {-------------------------------------------------------------------------
807   Build the Remote to Host Table from the Nodes table and demand input file
808   PARAMS: NONE
809   ASSUMPTIONS:  Host to Remote file has been created and is open
810   -------------------------------------------------------------------------}
811   procedure Tf_InterOfc.BuildRem2Hst;
812   Var
813     Fld     : String;
814     Clli    : String;
815     t_Demand: TTable;
816     bm      : TBookMark;
```

```
817     swtype  : String;
818     CLat,CLong,HLat,HLong: Double;
819     Distance: double;
820     TempInd : integer;
821   begin
822     try
823       {create the demand input table from the text file}
824       t_Demand := TTable.Create(Self);
825       with t_Demand do begin
826         if Active then
827           Close;
828         DatabaseName := 'modata';
829         TableName := 'IOTDEMND';
830         TableType := ttParadox;
831         Exclusive := True;
832         with FieldDefs do begin
833           Clear;
834           Add ('CLLI',    ftString, 11, False);
835           Add ('DS0SAL',  ftInteger, 0, False);
836           Add ('DS1SAL',  ftInteger, 0, False);
837           Add ('DS3SAL',  ftInteger, 0, False);
838           Add ('SwPorts', ftInteger, 0, False);
839         end;
840         with IndexDefs do begin
841           Clear;
842           Add('Main', 'CLLI',[ixPrimary, ixUnique]);
843         end;
844         CreateTable;
845         Open;
846       end;
847
848       {import the demand text file}
849       if Not ExtImportData(t_Demand, edUSDEMD.Text, nil, 1) then begin
850         t_Demand.Close;
851         exit;
852       end;
853
854       { open the Nodes table }
855       TabName :=  f_associate.GetFileName(2003);
```

```
856       With dm_ICM.t_Nodes do begin
857         if Active then Close;
858         DatabaseName := ExtractFilePath(TabName);
859         TableName := ExtractFilename(TabName);
860         TableType := ttDefault;
861         Exclusive := True;
862         Open;
863         First;
864
865         {build the H2R table from Nodes and demand tables}
866         TempInd:=0;  // temporary index variable
867         while not EOF do begin
868           swtype := fieldbyName('Switch Type').AsString;
869           if uppercase(Copy(swtype,1,3) ) = 'REM' then begin {if a remote clli}
870             dm_ICM.t_Hst2Rem.Append;
871             dm_ICM.t_Hst2Rem.FieldByName('HCLLI').AsString := FieldByName('Host CLLI').AsString;
872             dm_ICM.t_Hst2Rem.FieldByName('RCLLI').AsString := FieldByName('CLLI').AsString;
873             CLat  := FieldByName('Latitude').AsFloat;
874             CLong := FieldByName('Longitude').AsFloat;
875
876             {get the SAL values from the demand table}
877             Fld := dm_ICM.t_Nodes.FieldByName('CLLI').AsString;
878             t_Demand.First;
879             if t_Demand.FindKey([Fld]) then begin
880               dm_ICM.t_Hst2Rem.FieldByName('DS0SAL').AsInteger  := t_Demand.FieldByName('DS0SAL').AsInteger;
881               dm_ICM.t_Hst2Rem.FieldByName('DS1SAL').AsInteger  := t_Demand.FieldByName('DS1SAL').AsInteger;
882               dm_ICM.t_Hst2Rem.FieldByName('DS3SAL').AsInteger  := t_Demand.FieldByName('DS3SAL').AsInteger;
883               // RFPC0052a
884               dm_ICM.t_Hst2Rem.FieldByName('SWPorts').AsInteger := t_Demand.FieldByName('SWPorts').AsInteger;
885             end
886             else begin
887               MessageDlg('SAL Calc: Cannot locate Clli ['+Fld+'] in demand file.' , mtError, [mbOk], 0);
888               dm_ICM.t_Hst2Rem.FieldByName('DS0SAL').AsInteger := 0;
889               dm_ICM.t_Hst2Rem.FieldByName('DS1SAL').AsInteger := 0;
890               dm_ICM.t_Hst2Rem.FieldByName('DS3SAL').AsInteger := 0;
891               // RFPC0052a
892               dm_ICM.t_Hst2Rem.FieldByName('SWPorts').AsInteger := 0;
893             end;
894
```

```
895           {get Nodes Lat/Long for Host for distance calc}
896           bm := GetBookMark;
897           Fld := dm_ICM.t_Nodes.FieldByName('Host CLLI').AsString;
898           Clli := dm_ICM.t_Nodes.FieldByName('CLLI').AsString;
899           dm_ICM.t_Nodes.First;
900           if dm_ICM.t_Nodes.FindKey([Fld]) then begin
901              HLat  := FieldByName('Latitude').AsFloat;
902              HLong := FieldByName('Longitude').AsFloat;
903              Distance:=Ceil(CalcDistance(CLat,CLong,HLat,HLong));
904              // write out the record to the arc file with an "H2R" placeholder so
905              // that the TClli can be updated later when the H2H table has been
906              // created
907              TempInd:=TempInd+1;
908              WriteH2RArcFileRec('H2R',CLLI,Fld,CLat,CLong,HLat,HLong,TempInd,Round(Distance),t_IArcFile);
909           end
910           else begin
911              MessageDlg('Distance Calc: Cannot locate Host CLLI ['+FieldByName('Host CLLI').AsString+
912                        '] for ['+FieldByName('CLLI').AsString+'] in Nodes.', mtError, [mbOk], 0);
913           end;
914
915           dm_ICM.t_Hst2Rem.Post;
916           dm_ICM.t_Nodes.GoToBookmark(bm);  {return to Nodes record}
917         end; {if clli is a remote clli}
918         dm_ICM.t_Nodes.Next;
919       end; {do while not eof}
920     end; {do with t_Nodes}
921   finally
922     t_Demand.Close;
923     t_Demand.Free;
924     dm_ICM.t_Hst2Rem.Close;
925     dm_ICM.t_Hst2Rem.Open;
926   end;
927 end; {BuildHst2Rem}
928
929
930 {-------------------------------------------------------------------------
931 associate a new file with the table in the active notebook page
932 -------------------------------------------------------------------------}
933 procedure Tf_InterOfc.OpenClick(Sender: TObject);
```

```
934  var TabName: String;
935  begin
936    TabName := SelectFile('Select Table to Open','*.DB',
937                                    'Database|*.db;*.dbf',AliasPath);
938    if TabName = '' then exit;
939
940    if TabSet.TabIndex = 0 then begin
941      with dm_ICM.t_Hst2Hst do begin
942        Active := False;
943        DatabaseName := ExtractFilePath(H2HTabName);
944        TableName := ExtractFilename(TabName);
945        TableType := ttDefault;
946        Exclusive := True;
947        Open;
948      end;
949      WeAreWatching.LogEvent(mNORMAL,'OpenClick',' Open Host to Host Table: '+ ExtractFileName(TabName));
950    end
951    else if TabSet.TabIndex = 1 then begin
952      with dm_ICM.t_Hst2Rem do begin
953        Active := False;
954        DatabaseName := ExtractFilePath(H2HTabName);
955        TableName := ExtractFilename(TabName);
956        TableType := ttDefault;
957        Exclusive := True;
958        Open;
959      end;
960      WeAreWatching.LogEvent(mNORMAL,'OpenClick',' Open Host to Remote Table: '+ ExtractFileName(TabName));
961    end
962    else if TabSet.TabIndex = 2 then begin
963      with dm_ICM.t_Tandems do begin
964        Active := False;
965        DatabaseName := ExtractFilePath(TndTabName);
966        TableName := ExtractFilename(TabName);
967        TableType := ttDefault;
968        Exclusive := True;
969        Open;
970      end;
971      WeAreWatching.LogEvent(mNORMAL,'OpenClick',' Open Tandems Table: '+ ExtractFileName(TabName));
972    end
```

```
973     else if TabSet.TabIndex = 3 then begin
974       with dm_ICM.t_InterRing do begin
975         Active := False;
976         DatabaseName := ExtractFilePath(IRTabName);
977         TableName := ExtractFilename(TabName);
978         TableType := ttDefault;
979         Exclusive := True;
980         Open;
981       end;
982       WeAreWatching.LogEvent(mNORMAL,'OpenClick',' Open InterRing Table: '+ ExtractFileName(TabName));
983     end;
984     cb_Assoc.Checked := True;
985   end; {procedure Tf_InterOfc.OpenClick}
986
987   {-------------------------------------------------------------------------
988   Calculates the ADMOC48Comms and ADMOC12Comms for TndCompany <> 'O' and stores them
989   in the tandems table for lookup purposes during the investment calculations.
990
991   Assumptions: t_tandems, t_Nodes, t_Hst2Hst have been created and opened.
992   -------------------------------------------------------------------------}
993   procedure Tf_InterOfc.CalcADMOCComms(t_tandems: TTable);
994   var SonetEQ,pRing,Ring,i : integer;
995       Clli, HubClli : string;
996       SWPorts : double;
997       DS0SAL, DS1SAL, DS3SAL, ALinkDS0 : double;
998     function isGTEOwnedTandem(Clli: string): boolean;
999     var isGTEOwned : boolean;
1000    begin
1001      isGTEOwned:=false;
1002      with t_Tandems do
1003        begin
1004          if not Active then Open;
1005          IndexName:='';
1006          if FindKey([Clli]) and (FieldByName('TndCompany').AsString <> 'O') then
1007            isGTEOwned:=true
1008          else if not FindKey([Clli]) then
1009            begin
1010              IndexName:='byColoClli';
1011              if FindKey([Clli]) and (FieldByName('TndCompany').AsString <> 'O') then
```

```
1012                isGTEOwned:=true;
1013              IndexName:='';
1014            end;
1015        end;
1016      Result:=isGTEOwned;
1017    end; {isGTEOwnedTandem}
1018  begin
1019    // RFPC0052a  init the ADMOC arrays
1020    FillChar(ADMOC48Comms,SizeOf(ADMOC48Comms),0);
1021    FillChar(ADMOC12Comms,SizeOf(ADMOC12Comms),0);
1022    // clear out the hubclli list
1023    HubCllis.Clear;
1024    // set the index for TClli ranging
1025    dm_ICM.t_Hst2Hst.IndexName:='ByHubClli';
1026    // go to the top of the table
1027    dm_ICM.t_Hst2Hst.First;
1028
1029    // cycle through @record saving out the unique names to the string list
1030    Clli:='';
1031    with dm_ICM.t_Hst2Hst do
1032      while not eof do
1033        begin
1034          HubClli := FieldByName('HubClli').AsString;
1035          if (HubClli <> '') and (HubClli <> Clli) and isGTEOwnedTandem(HubClli) then
1036            begin
1037              HubCllis.Add(HubClli);
1038              Clli:=HubClli;
1039            end;
1040          Next;
1041        end; {while not eof}
1042
1043    // cycle through the HubClli list and compute the common equipment
1044    if (HubCllis.Count > 0) then
1045      begin
1046        for i:=1 to HubCllis.Count do
1047          begin
1048            ADMOC48Comms[i]:=0;  ADMOC12Comms[i]:=0;     //RFPC0052a
1049            HubClli:=HubCllis[i-1];  //RFPC0052a
1050            // get all the records in the H2H table for the specified TClli
```

```
1051            dm_ICM.t_Hst2Hst.SetRange([HubClli],[HubClli]);
1052            dm_ICM.t_Hst2Hst.First;
1053            pRing:=0;  SonetEQ:=0;
1054            // cycle through the h2h table and
1055            while not dm_ICM.t_Hst2Hst.EOF do
1056              begin
1057                // get all the necessary information
1058                Clli:=dm_ICM.t_Hst2Hst.FieldByName('ECLLI').AsString;
1059                Ring:=dm_ICM.t_Hst2Hst.FieldByName('Ring').AsInteger;
1060                DS0SAL:=dm_ICM.t_Hst2Hst.FieldByName('DS0SAL').AsFloat;
1061                DS1SAL:=dm_ICM.t_Hst2Hst.FieldByName('DS1SAL').AsFloat;
1062                DS3SAL:=dm_ICM.t_Hst2Hst.FieldByName('DS3SAL').AsFloat;
1063                SwPorts:=dm_ICM.t_Hst2Hst.FieldByName('SWPorts').AsInteger;
1064                if dm_ICM.t_SS7_SSP.FindKey([Clli]) then
1065                  ALinkDS0 := dm_ICM.t_SS7_SSP.FieldbyName('ALink').AsFloat
1066                else ALinkDS0 := 0.0;
1067                // check to see if a new ring was encountered
1068                if (pRing <> Ring) then
1069                  begin
1070                    if (SonetEq >= 12) then ADMOC48Comms[i]:=ADMOC48Comms[i]+1
1071                    else if (SonetEq >= 3) then ADMOC12Comms[i]:=ADMOC12Comms[i]+1;
1072                    SonetEq:=0;
1073                    pRing:=Ring;
1074                  end;
1075                SonetEQ := SonetEQ + Ceil(((SWPorts/28.0) *
1076                                  (IntraRingFactor/2)) +
1077                                  (((DS0SAL + ALinkDS0)  / 28.0) / 24.0) +
1078                                  (DS1SAL / 28.0) + DS3SAL);
1079                dm_ICM.t_Hst2Hst.Next;
1080              end;
1081            if (SonetEq >= 12) then ADMOC48Comms[i]:=ADMOC48Comms[i]+1
1082            else if (SonetEq >= 3) then ADMOC12Comms[i]:=ADMOC12Comms[i]+1;
1083          end;
1084      end; {if hubcllis count > 0}
1085
1086    // reset the table options
1087    with dm_ICM.t_Hst2Hst do begin
1088      CancelRange;
1089      IndexName:='';
```

```
1090        First;
1091      end;
1092    end; { Tf_InterOfc.CalcADMOCComms }
1093
1094    {-------------------------------------------------------------------------------
1095    For the InterRing, the ADMOC48Comm and ADMOC12Comm are computed and the
1096    investment calculations are performed outputing the IIADM and IIADMDS3
1097    components.  RFPC0067
1098    -------------------------------------------------------------------------------}
1099    procedure Tf_InterOfc.CalcInterRingADMInvestments;
1100    var TabName,HubClli,Clli  : string;
1101        SonetEQ,pRing,Ring,i : integer;
1102        ISWPorts : double;
1103        IIADMOC48Comm,IIADMOC12Comm,IIADMOC3Comm,OC3Card,OC12Card : integer;
1104        DSX1Max,DSX3Max : double;
1105      // compute the ADM common equipment
1106      procedure CalcADMOCs(i: integer);
1107      begin
1108        pRing:=0;  SonetEQ:=0;
1109        dm_ICM.t_InterRing.SetRange([HubClli],[HubClli]);
1110        dm_ICM.t_InterRing.First;
1111        while not dm_ICM.t_InterRing.eof do
1112          begin
1113            Clli:=dm_ICM.t_InterRing.FieldByName('IClli').AsString;
1114            Ring:=dm_ICM.t_InterRing.FieldByName('IRing').AsInteger;
1115            ISWPorts:=dm_ICM.t_InterRing.FieldByName('ISWPorts').AsInteger;
1116            // check to see if a new ring was encountered
1117            if (pRing <> Ring) then
1118              begin
1119                // update the ADMOC counts
1120                if (SonetEq >= 12) then IIADMOC48Comms[i]:=IIADMOC48Comms[i]+1
1121                else if (SonetEq >= 3) then IIADMOC12Comms[i]:=IIADMOC12Comms[i]+1;
1122                SonetEq:=0;
1123                pRing:=Ring;
1124              end;
1125            // compute the sonet equipment
1126            SonetEQ := SonetEQ + Ceil(ISWPorts/28.0/2);
1127            dm_ICM.t_InterRing.Next;
1128          end; {while not eof}
```

```
1129      // update the ADMOC counts
1130      if (SonetEq >= 12) then IIADMOC48Comms[i]:=IIADMOC48Comms[i]+1
1131      else if (SonetEq >= 3) then IIADMOC12Comms[i]:=IIADMOC12Comms[i]+1;
1132    end; {CalcADMOCs}
1133
1134    // Compute the investments and output the copmonents to the ActH2H table
1135    procedure CalcADMInvestment(Clli: string);
1136    var iADMvs, iADM, LabInvest, DSX3DS0 : double;
1137        SOC3Card,SOC12Card : integer;
1138    begin
1139      // get the switch ports
1140      ISWPorts:=dm_ICM.t_InterRing.FieldByName('ISWPorts').AsInteger;
1141
1142      {default if OC48 and OC12 counts are zero}
1143      if (IIADMOC48Comm = 0) and (IIADMOC12Comm = 0) and (IIADMOC3Comm = 0) then
1144        begin
1145          if SonetEQ >= 12 then IIADMOC48Comm := 1
1146          else if (SonetEq < 3) then IIADMOC3Comm := 1
1147          else IIADMOC12Comm := 1;
1148        end;
1149
1150      // compute the number of cards
1151      SOC3Card  := 0;
1152      SOC12Card := 0;
1153      if (IIADMOC48Comm > 0) then SOC12Card := 2* Ceil(ISWPorts / OC12Card)
1154      else if (IIADMOC12Comm > 0) then SOC3Card := 2* Ceil(ISWPorts / OC3Card)
1155      else SOC3Card := 2 * IIADMOC3Comm;
1156
1157      DSX3DS0  := (24.0 * ISWPorts);
1158
1159      // compute the investments
1160      if (IIADMOC48Comm > 0) then begin
1161        iADM := (f_OSPMaterial.Mat('ADM', 48)*IIADMOC48Comm) +
1162                (f_OSPMaterial.Mat('OC',  12)*SOC12Card);
1163        LabInvest := (f_OSPLabor.Labor('LADM48','Rural Rate')*IIADMOC48Comm) +
1164                     (f_OSPLabor.Labor('LOC12CARD','Rural Rate')*SOC12Card);
1165        iADMvs  := (DSX3DS0 /28.0) * ((iADM + LabInvest) / (IIADMOC48Comm * 48.0 * 28.0)) +   //RFPC0032
1166                   (f_OSPMaterial.Mat('OC', 12) +                                           //RFPC0032
1167                    f_OSPLabor.Labor('LOC12CARD','Rural Rate')) / OC12Card; //RFPC0032
```

```
1168        end
1169      else if (IIADMOC12Comm > 0) then begin
1170        iADM := (f_OSPMaterial.Mat('ADM', 12)*IIADMOC12Comm) +
1171                (f_OSPMaterial.Mat('OC',   3)*SOC3Card);
1172        LabInvest := (f_OSPLabor.Labor('LADM12','Rural Rate')*IIADMOC12Comm) +
1173                     (f_OSPLabor.Labor('LOC3CARD','Rural Rate')*SOC3Card);
1174        iADMvs  := (DSX3DS0 /28.0) * ((iADM + LabInvest) / (IIADMOC12Comm * 12.0 * 28.0)) +   //RFPC0032
1175                   (f_OSPMaterial.Mat('OC', 3) +                          //RFPC0032
1176                    f_OSPLabor.Labor('LOC3CARD','Rural Rate')) / OC3Card; //RFPC0032
1177        end
1178      else begin
1179        iADM := (f_OSPMaterial.Mat('ADM', 3)*IIADMOC3Comm) +
1180                (f_OSPMaterial.Mat('OC',  3)*SOC3Card);
1181        LabInvest := (f_OSPLabor.Labor('LADM3','Rural Rate')*IIADMOC3Comm) +
1182                     (f_OSPLabor.Labor('LOC3CARD','Rural Rate')*SOC3Card);
1183        iADMvs  := (DSX3DS0 /28.0) * ((iADM + LabInvest) / (IIADMOC3Comm * 3.0 * 28.0)) +   //RFPC0032
1184                   (f_OSPMaterial.Mat('OC', 3) +                          //RFPC0032
1185                    f_OSPLabor.Labor('LOC3CARD','Rural Rate')) / OC3Card; //RFPC0032
1186      end;
1187      // write ADM records
1188      if (dm_ICM.t_InterRing.FieldByName('IClli Substitution').AsString <> '') then
1189        Clli:=dm_ICM.t_InterRing.FieldByName('IClli Substitution').AsString;
1190      if (iADM + LabInvest) > 0 then begin
1191        IOF_Table.Write(Clli, '223200', 'IIADM', 'I',
1192              (iADM + LabInvest), iADMvs, (iADM + LabInvest)-iADMvs, false);  //RFPC0032
1193
1194        IOF_Table.Write(Clli, '223200', 'IIADMDS1', 'I',
1195             ISWPorts, ((IIADMOC48Comm*48)+(IIADMOC12Comm*12)+(IIADMOC3Comm*3))*28, ISwPorts, false);
1196  //RFPC0032
1197      end;
1198    end; {CalcADMInvestment}
1199  begin
1200    FillChar(IIADMOC48Comms,SizeOf(IIADMOC48Comms),0);
1201    FillChar(IIADMOC12Comms,SizeOf(IIADMOC12Comms),0);
1202    FillChar(ISWPorts,SizeOf(ISWPorts),0);
1203    // clear out the ihubclli, ihubclli substitution, iclli and iclli substitution list
1204    iHubCllis.Clear;
1205    iHubCllisSub.Clear;
1206    iCllis.Clear;
```

```
1207    iCllisSub.Clear;
1208    iCllisHub.Clear;
1209    {Set Max settings for equipment calculation}
1210    DSX1Max      := Trunc(32*iADMIN_FILL);
1211    DSX3Max      := Trunc(10*iADMIN_FILL);
1212    OC3Card      := 84;
1213    OC12Card     := 336;
1214
1215    { Cycle through the InterRing Table }
1216    TabName :=  f_associate.GetFileName(3012);
1217    With dm_ICM.t_InterRing do
1218      begin
1219        { specify the table properties }
1220        if Active then Close;
1221        DatabaseName := ExtractFilePath(TabName);
1222        TableName := ExtractFilename(TabName);
1223        IndexName := 'ByHubClli';
1224        TableType := ttDefault;
1225        Exclusive := False;
1226        Open;
1227
1228        // cycle through @record saving out the unique names to the string list
1229        Clli:='';
1230        while not eof do
1231          begin
1232            HubClli := FieldByName('IHubClli').AsString;
1233            // add the iclli to the iclli list, along with the substitution and switch ports
1234            iCllis.Add(FieldByName('IClli').AsString);
1235            iCllisSub.Add(FieldByName('IClli Substitution').AsString);
1236            iCllisHub.Add(FieldByName('IHubClli').AsString);
1237            IISWPorts[iCllis.Count]:=FieldByName('ISWPorts').AsInteger;
1238            // add the unique hubclli to the ihubclli list, along with the substitution
1239            if (HubClli <> '') and (HubClli <> Clli) then
1240              begin
1241                iHubCllis.Add(HubClli);
1242                iHubCllisSub.Add(FieldByName('IHubClli Substitution').AsString);
1243                Clli:=HubClli;
1244              end;
1245            Next;
```

```
1246          end; {while not eof}
1247
1248        if (iHubCllis.Count > 0) then
1249          begin
1250            // cycle through @Hubclli, compute the ADMOC equipment, and the investment
1251            for i:=1 to iHubCllis.Count do
1252              begin
1253                HubClli:=iHubCllis[i-1];
1254                // reset the counts
1255                IIADMOC48Comms[i]:=0;  IIADMOC12Comms[i]:=0;
1256                IIADMOC48Comm:=0;  IIADMOC12Comm:=0;  IIADMOC3Comm:=0;
1257                // get the common equipment
1258                CalcADMOCs(i);
1259                // make sure the counts are reassigned
1260                IIADMOC48Comm:=IIADMOC48Comms[i];
1261                IIADMOC12Comm:=IIADMOC12Comms[i];
1262                IIADMOC3Comm:=0;
1263
1264                First;  // reset back to the first record
1265                while not eof do
1266                  begin
1267                    Clli:=FieldByName('IClli').AsString;
1268                    // compute and output the investments
1269                    CalcADMInvestment(Clli);
1270                    Next;
1271                  end;
1272                // now compute the investment
1273              end; {for i loop}
1274          end; {if count > 0}
1275
1276        // close the table
1277        Close;
1278      end; {with dm_ICM.t_InterRing}
1279  end; {CalcInterRingADMInvestments}
1280
1281  {-------------------------------------------------------------------------
1282  Global Process procedure, called during report creation
1283  Process the hst2hst/hst2rem table and determine investment costs for equipment,
1284  labor and placement (distribution).
```

```
1285  ------------------------------------------------------------------------}
1286  procedure Tf_InterOfc.Process;
1287  var TabName : String;
1288      I: Integer;
1289  begin
1290    Mode := 2;
1291    WeAreWatching.LogEvent(mNORMAL,'Process','');  // blank line
1292    WeAreWatching.LogEvent(mNORMAL,'Process','Processing Interoffice');
1293    CancelProcess := False;
1294
1295    SetGridConstants;              {set program global variables}
1296
1297    //get runtime option settings
1298    if t_IOFSet.Active = False then
1299      t_IOFSet.Open;
1300    iADMIN_FILL := t_IOFSet.FieldByName('Admin Fill').AsFloat;
1301    Asmp.Air2Route    := t_IOFSet.FieldByName('Air2Route').AsFloat;
1302    IntraRingFactor := t_IOFSet.FieldByName('IntraRing Factor').AsFloat;
1303    if (t_IOFSet.FieldByName('Aerial Span').AsString <> '') then
1304      Asmp.Aer_Fi_Span := t_IOFSet.FieldByName('Aerial Span').AsFloat;
1305    if (t_IOFSet.FieldByName('Buried Span').AsString <> '') then
1306      Asmp.Bur_Fi_Span := t_IOFSet.FieldByName('Buried Span').AsFloat;
1307
1308    // validate variable settings
1309    if iADMIN_FILL = 0 then begin
1310      iADMIN_FILL := 1;
1311      WeAreWatching.LogEvent(mVALID,'Process','Option Validation: Admin Fill is set to 0, changed to 1');
1312    end;
1313    if IntraRingFactor = 0 then begin
1314      IntraRingFactor := 2;  // ultimately divided by 2
1315      WeAreWatching.LogEvent(mVALID,'Process','Option Validation: IntraRing Factor is set to 0, changed to
1316  2');
1317    end;
1318
1319    WeAreWatching.LogEvent(mNORMAL,'Process','RunTime Setting: ADMIN_FILL: '+ FloatToStr(iADMIN_FILL));
1320    WeAreWatching.LogEvent(mNORMAL,'Process','RunTime Setting: Intra-Ring Factor:
1321  '+FloatToSTr(IntraRingFactor));
1322    WeAreWatching.LogEvent(mNORMAL,'Process','RunTime Setting: Pole Spacing:
1323  '+FloatToSTr(Asmp.Pole_Spacing));
```

```
1324    WeAreWatching.LogEvent(mNORMAL,'Process','RunTime Setting: Manhole Spacing:
1325  '+FloatToStr(Asmp.Manhole_Spacing));
1326    { Host to Host Table }
1327    TabName :=  f_associate.GetFileName(3007);
1328    With dm_ICM.t_Hst2Hst do begin
1329        if Active then Close;
1330        DatabaseName := ExtractFilePath(TabName);
1331        TableName := ExtractFilename(TabName);
1332        IndexFieldNames := '';
1333        TableType := ttDefault;
1334        Exclusive := False;
1335        Open;
1336    end;
1337
1338    { Host to Remote Table }
1339    TabName :=  f_associate.GetFileName(3008);
1340    With dm_ICM.t_Hst2Rem do begin
1341        if Active then Close;
1342        DatabaseName := ExtractFilePath(TabName);
1343        TableName := ExtractFilename(TabName);
1344        IndexFieldNames := '';
1345        TableType := ttDefault;
1346        Exclusive := False;
1347        Open;
1348    end;
1349
1350    {open up supporting files}
1351    {Nodes table}
1352    TabName :=  f_associate.GetFileName(2003);
1353    With dm_ICM.t_Nodes do begin
1354      if Active then Close;
1355      DatabaseName := ExtractFilePath(TabName);
1356      TableName := ExtractFilename(TabName);
1357      TableType := ttDefault;
1358      Exclusive := False;
1359      Open;
1360    end;
1361
1362    {switch table for Host/Rem lookup}
```

```
1363    TabName :=  f_associate.GetFileName(2001);
1364    With dm_ICM.t_Switch do begin
1365      if Active then Close;
1366      DatabaseName := ExtractFilePath(TabName);
1367      TableName := ExtractFilename(TabName);
1368      TableType := ttDefault;
1369      Exclusive := False;
1370      IndexName:='';
1371      Open;
1372    end;
1373
1374    TabName :=  f_associate.GetFileName(2010); //RFPC0032 State SSP table
1375    With dm_ICM.t_SS7_SSP do
1376      begin
1377         if Active then Close;
1378         DatabaseName := ExtractFilePath(TabName);
1379         TableName := ExtractFilename(TabName);
1380         IndexFieldNames := '';
1381         IndexName := '';
1382         TableType := ttDefault;
1383         Exclusive := True;
1384         Open;
1385      end;
1386
1387    { tandem to tandem table for ADMOC calculations}
1388    TabName :=  f_associate.GetFileName(3011); //RFPC0052a Tandem to Tandem table
1389    with dm_ICM.t_Tandems do begin
1390      if Active then Close;
1391      DatabaseName := ExtractFilePath(TabName);
1392      TableName := ExtractFilename(TabName);
1393      TableType := ttParadox;
1394      Exclusive := False;
1395      Open;
1396    end;
1397
1398    if dm_ICM.t_Nodes.FieldByName('Tier B').AsBoolean
1399      then Tier_Flag := 'B'
1400      else Tier_Flag := 'A';
1401
```

```
1402    try
1403      {process the host to host}
1404      IOF_Table.Open;
1405      dm_ICM.t_Hst2Hst.Active := False;
1406      dm_ICM.t_Hst2Hst.IndexName := '';
1407      dm_ICM.t_Hst2Hst.Active := True;
1408      dm_ICM.t_Hst2Hst.First;
1409      WeAreWatching.LogEvent(mNORMAL,'Process','');
1410      WeAreWatching.LogEvent(mNORMAL,'Process',' Processing Host to Host.');
1411
1412      // init the progress bar
1413      f_Demand.Gauge1.Progress := 0;
1414      f_Demand.l_Progress.Caption := ' Processing Host to Host ';
1415      Application.ProcessMessages;
1416
1417      dm_ICM.t_Hst2Hst.DisableControls;
1418
1419      // compute the ADMOC48Comms and ADMOC12Comms
1420      HubClli := TStringList.Create;    // first instantiate the hubclli list
1421      CalcADMOCComms(dm_ICM.t_Tandems);
1422      // RFPC0067 calculate the InterRing ADM investments
1423      iHubClli := TStringList.Create;    // instantiate the interring unique ihubclli list
1424      iHubCliSub := TStringList.Create; // instantiate the interring ihubclli substitution list
1425      iClli := TStringList.Create;       // instantiate the interring iclli list
1426      iCliSub := TStringList.Create;     // instantiate the interring iclli substitution list
1427      iCliHub := TStringList.Create;     // instantiate the interring ihubclii list
1428      CalcInterRingADMInvestments;
1429
1430
1431      {create the log file for reviewing ring distance calculations}
1432      AssignFile(tFile, 'HostDist.log');
1433      ReWrite(tFile);  i:=0;
1434      while not dm_ICM.t_Hst2Hst.Eof do begin
1435        if CancelProcess then         {check for a cancel}
1436          break;
1437        i:=i+1;
1438        // update the progress bar
1439        f_Demand.Gauge1.Progress := Round(i / dm_ICM.t_Hst2Hst.RecordCount * 100.0);
1440        f_Demand.l_Progress.Caption := Format('H2H %d of %d', [i, dm_ICM.t_Hst2Hst.RecordCount]);
```

```
1441        Application.ProcessMessages;
1442
1443        WeAreWatching.LogEvent(mFUNC,'Process',' CalcCliInvest for ('+
1444  dm_ICM.t_Hst2Hst.FieldByName('ECLLI').AsString +', H, '+
1445                              dm_ICM.t_Hst2Hst.FieldByName('Ring').AsString+')');
1446        CalcCliInvest(dm_ICM.t_Hst2Hst, dm_ICM.t_Tandems, dm_ICM.t_Hst2Hst.FieldByName('ECLLI').AsString,
1447                      'H',dm_ICM.t_Hst2Hst.FieldByName('Ring').AsInteger);
1448        WeAreWatching.LogEvent(mFUNC,'CalcCliInvest','');
1449        dm_ICM.t_Hst2Hst.Next;
1450      end;
1451      CloseFile(tFile);    // close the distance calc log file
1452      dm_ICM.t_Hst2Hst.EnableControls;
1453
1454      {process the host to remotes}
1455      dm_ICM.t_Hst2Rem.Active := False;
1456      dm_ICM.t_Hst2Rem.IndexName := '';
1457      dm_ICM.t_Hst2Rem.Active := True;
1458      dm_ICM.t_Hst2Rem.First;
1459      WeAreWatching.LogEvent(mNORMAL,'Process','');
1460      WeAreWatching.LogEvent(mNORMAL,'Process',' Processing Host to Remote.');
1461      f_InterOfc.Caption := 'InterOffice Transport ['+dm_ICM.t_Hst2Rem.TableName+']';
1462
1463      f_Demand.Gauge1.Progress := 0;
1464      f_Demand.l_Progress.Caption := ' Processing Host to Remote ';
1465      Application.ProcessMessages;
1466      dm_ICM.t_Hst2Rem.DisableControls;
1467      i:=0;
1468      while not dm_ICM.t_Hst2Rem.Eof do begin
1469        if CancelProcess then          {check for a cancel}
1470          break;
1471        i:=i+1;
1472        // update the progress bar
1473        f_Demand.Gauge1.Progress := Round(i / dm_ICM.t_Hst2Rem.RecordCount * 100.0);
1474        f_Demand.l_Progress.Caption := Format('H2R %d of %d', [i, dm_ICM.t_Hst2Rem.RecordCount]);
1475        Application.ProcessMessages;
1476
1477        WeAreWatching.LogEvent(mFUNC,'Process',' CalcCliInvest for '+
1478  dm_ICM.t_Hst2Rem.FieldByName('RCLLI').AsString +'R'+ '0');
```

```
1479        CalcCliiInvest(dm_ICM.t_Hst2Rem, dm_ICM.t_Tandems, dm_ICM.t_Hst2Rem.FieldByName('RCLLI').AsString,
1480    'R',0);
1481        WeAreWatching.LogEvent(mFUNC,'Process','  Adding Host
1482    ('+dm_ICM.t_Hst2Rem.FieldByName('HCLLI').AsString+
1483                              ') Investment for '+dm_ICM.t_Hst2Rem.FieldByName('RCLLI').AsString);
1484        WeAreWatching.LogEvent(mFUNC,'CalcCliiInvest','');
1485        dm_ICM.t_Hst2Rem.Next;
1486      end;
1487      dm_ICM.t_Hst2Rem.EnableControls;
1488
1489      // Output the IS45 components for all the 45 offices that have collocated
1490      // offices.  Cycle through the tandems table, pick up the records that have
1491      // coloclli populated.
1492      with dm_ICM.t_Tandems do
1493        begin
1494          if not Active then Open;    // open the table if it is closed
1495          IndexName:='';              // reset the index name
1496
1497          // cycle through the entire table, processing only the populated coloclli records
1498          First;
1499          while not eof do
1500            begin
1501              // process the populated coloclli records only
1502              if (FieldByName('ColoClli').AsString <> '') then
1503                IOF_Table.Write(FieldByName('ColoClli').AsString, '000000', 'IS45', 'I',1,1,1,false);
1504              Next;
1505            end; {while not eof}
1506        end;
1507
1508      f_InterOfc.Height := 451;
1509
1510    finally  {cleanup and exit}
1511      HubCllis.Free;
1512      iHubCllis.Free;
1513      iHubCllisSub.Free;
1514      iCllis.Free;
1515      iCllisSub.Free;
1516      iCllisHub.Free;
1517      dm_ICM.t_Nodes.CancelRange;
```

```
1518      t_Grid.Close;
1519      t_IOFSet.Close;
1520      dm_ICM.t_Switch.Close;
1521      dm_ICM.t_SS7_SSP.Close;
1522      if dm_ICM.t_Tandems.Active then dm_ICM.t_Tandems.Close;
1523      if dm_ICM.t_InterRing.Active then dm_ICM.t_InterRing.Close;
1524      btnCancel.Caption := 'Done';
1525    end; {try-finally}
1526  end; {procedure Tf_InterOfc.Process}
1527
1528  function Tf_InterOfc.IsColocated(Clli: string; t_tandems: TTable): boolean;
1529  var found: boolean;
1530  begin
1531    if not t_tandems.Active then t_tandems.Open;
1532    t_tandems.IndexName:='ByColoClli';
1533    Result:=t_tandems.FindKey([Clli]);
1534    t_tandems.IndexName:='';
1535  end;
1536
1537  {-------------------------------------------------------------------------
1538  COMBINED FOR BOTH H2H AND H2R
1539  Calculate the equipment, engineering/labor and Placement Investment
1540     for a given CLLI
1541  -------------------------------------------------------------------------}
1542  procedure Tf_InterOfc.CalcCliInvest(t_IOF, t_tandems:tTable; CliCode:String;
1543                                      FacType: Char; CRing: Integer);
1544  var  TndCLLI: String;
1545       {equip variables}
1546       SonetEQ,  DSX1Jump, DSX1Comm, DSX3Jump, DSX3Comm,
1547       RemoteEq, SDS3Card, SDS1Card, FDPMod, FDPComm, DCS10Comm, CBComm,
1548       CBCard, DCS11Prt, DCS31Comm, DCS31Prt, DCS33Prt, Mux31Comm, RemoteSW,
1549       SWPorts, DS0SAL, DS1SAL, DS3SAL, iDSX1, iDSX3, iFDP, iCB, iDCS10, iDCS31,
1550       iMux31, iADM, ISWPorts: Double;
1551       ALinkDS0, DSX3DS0, DSX1DS0, DCS10DS0, iADMvs, iDSX1vs, iDSX3vs, //RFPC0032
1552       iDCS10vs, iDCS31vs, iMUX31vs, iFDPvs, iCBvs  : Double; //RFPC0032
1553       DSX1Max, DSX3Max, CBMax, DCS10DS1Max, Prt11Max, Prt31Max,
1554       SonetADM, Prt33Max, DCS31DS1Max, DCS31DS3Max, DS1PCard, DS3PCard: Integer; //RFPC0032
1555
1556       {investment and calculation variables}
```

```
1557        EqInvest, LabInvest, PlaceInvest, tLabInvest, dValue,
1558        Aerial, Buried, Undgnd: Double;
1559        HostInvst,RecCount: Integer;
1560
1561        {account variables}
1562        EquipActNo: String;
1563
1564        SwType: string;      {switch type}
1565        bh: String;          {bnf header string  IR:Remote or IH:Host}
1566        mStr: String[80];
1567        bmClli: TBookMark;  {bookmark to return to}
1568        TabName : String; //RFPC0032 Table name
1569        writeRemotes : boolean;
1570
1571        {OC48, OC12 and OC3 counts}
1572        ADMOC48Comm, ADMOC12Comm, ADMOC3Comm : integer;
1573        IIADMOC48Comm, IIADMOC12Comm : integer;      //RFPC0067
1574        i: integer;
1575        HFactor, IFactor: double;  //RFPC0067
1576        HubClli: string;  //RFPC0067
1577    begin
1578      if dm_ICM.t_SS7_SSP.FindKey([CliiCode])
1579         then begin
1580                ALinkDS0 := dm_ICM.t_SS7_SSP.FieldbyName('ALink').AsFloat;
1581              end
1582         else begin
1583                ALinkDS0 := 0.0;
1584              end;
1585
1586      {procedure setup}
1587      LabInvest := 0;
1588      RecCount := 0;
1589
1590      if FacType = 'R' then
1591        HostInvst   := 2     {double investment for host side}
1592      else HostInvst := 1;
1593
1594      writeRemotes:=(FacType='H');
1595
```

```
1596    {set the account numbers}
1597    EquipActNo  := '223200';   {all equipment}
1598
1599    {define the bnf code's header}
1600    if FacType = 'R' then bh := 'IR';  {header for remote bnf record}
1601    if FacType = 'H' then bh := 'IH';  {header for host bnf record}
1602
1603    {Set Max settings for equipment calculation}
1604    DSX1Max     := Trunc(32*iADMIN_FILL);
1605    DSX3Max     := Trunc(10*iADMIN_FILL);
1606    CBMax       := Trunc(48*iADMIN_FILL);  //RFPC TBD
1607    DCS10DS1Max := Trunc(128*iADMIN_FILL);
1608    Prt11Max    := 8;
1609    Prt33Max    := 1;
1610    Prt31Max    := 4;
1611    DCS31DS1Max := Trunc(448*iADMIN_FILL);
1612    DCS31DS3Max := Trunc(64*iADMIN_FILL);
1613    DS1PCard    := 14;
1614    DS3PCard    := 3;
1615
1616    dm_ICM.t_Nodes.FindKey([ClliCode]);
1617
1618    {if no traffic is associated with this CLLI, then don't process}
1619    if (t_IOF.FieldByName('SWPorts').AsInteger < 1) and
1620       (dm_ICM.t_Nodes.FieldbyName('class').AsInteger <> 4) then begin
1621      WeAreWatching.LogEvent(mERROR,'CalcClliInvest', ClliCode+': No Traffic at node');
1622      exit;
1623    end
1624    else if (dm_ICM.t_Nodes.FieldbyName('class').AsInteger <> 5) then begin
1625      WeAreWatching.LogEvent(mERROR,'CalcClliInvest', ClliCode+': Class 4 or 4/5 tandem, do not process.');
1626      exit;
1627    end;
1628
1629    {init the OC48, OC12 and OC3 counts}
1630    ADMOC48Comm:=0;    ADMOC12Comm:=0;    ADMOC3Comm:=0;
1631    IIADMOC48Comm:=0;  IIADMOC12Comm:=0;  ISWPorts:=0;
1632
1633    {if H2H set record range for concentrator ring
1634     by tandem's clli code and ring number}
```

```
1635      SonetEQ := 0;
1636     if FacType = 'H' then begin
1637       bmClli := t_IOF.GetBookMark;
1638       TndCLLI := t_IOF.FieldByName('HubClli').AsString;
1639       t_IOF.Active := False;
1640       t_IOF.IndexName := 'ByHubClli';
1641       t_IOF.Active := True;
1642       t_IOF.First;
1643       with t_IOF do begin
1644         CancelRange;
1645         SetRange([TndClli,CRing],[TndClli,CRing]);
1646         First;
1647       end;
1648       Refresh;
1649
1650       {get the OC48 and OC12 counts from the common equipment array}
1651       if (CliCode=TndClli) then
1652         begin
1653           i := HubCllis.IndexOf(TndClli);
1654           if (i >= 0) then
1655             begin
1656               ADMOC48Comm:=ADMOC48Comms[i+1];
1657               ADMOC12Comm:=ADMOC12Comms[i+1];
1658             end;
1659           // get the switch ports, first lookup the name in the iclli list
1660           i := iCllis.IndexOf(TndClli);
1661           // check the substitution list if not found in the iclli list
1662           if (i < 0) then i := iCllisSub.IndexOf(TndClli);
1663           if (i >= 0) then
1664             begin
1665               ISWPorts:=IISWPorts[i+1];
1666               HubClli:=iCllisHub[i];
1667               // get the InterRing related common equipment
1668               i := iHubCllis.IndexOf(HubClli);
1669               // check the substitution list if not found in the ihubclli list
1670               if (i < 0) then i := iHubCllisSub.IndexOf(HubClli);
1671               // if found then assign the common equipment totals
1672               if (i >= 0) then
1673                 begin
```

```
1674                  IIADMOC48Comm:=IIADMOC48Comms[i+1];
1675                  IIADMOC12Comm:=IIADMOC12Comms[i+1];
1676                end;
1677              end;
1678          end;
1679
1680        { get the sonet equipment requirements for the ring}
1681        with t_IOF do begin
1682          while not EOF do begin
1683            Application.ProcessMessages;
1684            SonetEQ := SonetEQ + Ceil(((FieldByName('SWPorts').AsFloat/28.0) *        //RFPC0032
1685                                (IntraRingFactor/2)) +                                //RFPC0032
1686                                (((FieldByName('DS0SAL').AsFloat + ALinkDS0)  / 28.0) / 24.0) +   //RFPC0032
1687                                ((FieldByName('DS1SAL').AsFloat) / 28.0) +            //RFPC0032
1688                                FieldByName('DS3SAL').AsFloat);                       //RFPC0032
1689            Inc(RecCount);
1690            t_IOF.Next;
1691          end;
1692        end;
1693        {cleanup}
1694        t_IOF.CancelRange;
1695        t_IOF.Active := False;
1696        t_IOF.IndexName := '';
1697        t_IOF.Active := True;
1698        t_IOF.First;
1699        t_IOF.GotoBookmark(bmClli);  {return to clli record}
1700        t_IOF.FreeBookMark(bmClli);
1701      end {if host to host}
1702      else begin {if host to remote}
1703       {assign placement variables}
1704
1705        SonetEQ := Ceil(((t_IOF.FieldByName('SWPorts').AsFloat/28.0) *        //RFPC0032
1706                    (IntraRingFactor/2)) +                                    //RFPC0032
1707                    (((t_IOF.FieldByName('DS0SAL').AsFloat + ALinkDS0)  / 28.0) / 24.0) +   //RFPC0032
1708                    ((t_IOF.FieldByName('DS1SAL').AsFloat) / 28.0) +//RFPC0032
1709                    t_IOF.FieldByName('DS3SAL').AsFloat);                     //RFPC0032
1710      end;  {if host to remote}
1711
1712      {Output specific Hosts distance bnf's}
```

```
1713    if FacType = 'H' then begin
1714      {output number of nodes on ring}
1715      IOF_Table.Write(CliiCode,'000000','IHNODES','I',RecCount,RecCount,RecCount,writeRemotes);
1716    end;
1717
1718   {assign table values to variables for clarity}
1719    with t_IOF do begin
1720      if FacType = 'H' then
1721        RemoteSW := FieldByName('RemoteSW').AsInteger;  {only for Tandem}
1722      SWPorts  := FieldByName('SWPorts').AsInteger;
1723      DS0SAL   := FieldByName('DS0SAL').AsInteger;
1724      DS1SAL   := FieldByName('DS1SAL').AsInteger;
1725      DS3SAL   := FieldByName('DS3SAL').AsInteger;
1726    end;
1727
1728    {Size the clli's equipment to traffic requirements}
1729    if SonetEQ >= 12 then SonetADM := 48
1730    else if SonetEQ >= 3 then SonetADM := 12
1731    else SonetADM := 3;
1732
1733    {default if OC48 and OC12 counts are zero}
1734    if (ADMOC48Comm = 0) and (ADMOC12Comm = 0) and (ADMOC3Comm = 0) then
1735      begin
1736        if SonetEQ >= 12 then ADMOC48Comm := 1
1737        else if (FacType='R') and (SonetEq < 3) then ADMOC3Comm:=1
1738        else ADMOC12Comm := 1;
1739      end;
1740
1741    {determine the amount of fiber distribution modules and panels required}
1742    if FacType = 'H' then begin  {if host, then size by traffic}
1743      FDPMod   := (2*(ADMOC48Comm+ADMOC12Comm+IIADMOC48Comm+IIADMOC12Comm))+RemoteSW;
1744      FDPComm  := Ceil(((2*(ADMOC48Comm+ADMOC12Comm+IIADMOC48Comm+IIADMOC12Comm))+RemoteSW)/8);
1745    end
1746    else begin                    {if remote, set to one}
1747      FDPMod  := 1;
1748      FDPComm := 1;
1749    end;
1750
1751    {set the remaining sizing variables for investment calculations}
```

```
1752    {  RFPC0032
1753       DSX-1 Cross Connect:
1754
1755        DSX-1 Jumpers = SWPORTS + DS1SAL + Ceil((DS0SAL + ALinkDS0) / 24)
1756        DSX-1 Common Equipment = Ceil((SWPORTS + DS1SAL
1757                                  + Ceil((DS0SAL + ALinkDS0) / 24))/DSX1Max)
1758
1759       DSX-3 Cross Connect:
1760
1761        DSX-3 Modules =  DS3SAL + Ceil((SWPORTS + DS1SAL
1762                              + Ceil((DS0SAL + ALinkDS0)/24))/28)
1763        DSX-3 Common Equipment = Ceil((DS3SAL + Ceil((SWPORTS + DS1SAL
1764                                 + Ceil((DS0SAL + ALinkDS0)/24))/28))/DSX3Max)
1765
1766    }
1767    RemoteEq := RemoteSW;
1768    DSX1Jump := SWPorts + DS1SAL + Ceil((DS0SAL + ALinkDS0)/24);                    // RFPC0032
1769    DSX1Comm := Ceil((SWPorts + DS1SAL + Ceil((DS0SAL + ALinkDS0)/24))/DSX1Max);    // RFPC0032
1770    DSX3Jump := DS3SAL + Ceil((SWPorts + DS1SAL + Ceil((DS0SAL + ALinkDS0)/24))/28); // RFPC0032
1771    DSX3Comm := Ceil((DS3SAL + Ceil((SWPorts + DS1SAL + Ceil((DS0SAL + ALinkDS0)/24))// RFPC0032
1772                /28))/DSX3Max);                                                     // RFPC0032
1773
1774    {determine quantity for each piece of equipment}
1775    if ((DS0SAL+ALinkDS0) < 1) then
1776      CBComm := 0
1777    else
1778      CBComm := Ceil((DS0SAL+ALinkDS0)/CBMax);  // RFPC0032
1779
1780    if ((DS0SAL+ALinkDS0) < 1) then
1781      CBCard := 0
1782    else
1783      CBCard := (DS0SAL+ALinkDS0);
1784
1785    if (FacType = 'H') and (DS1SAL > 0) and (dm_ICM.t_nodes.fieldbyName('Class').AsInteger = 5) and
1786       IsColocated(CLliCode,t_tandems) then begin
1787     { RFPC0032
1788       DCS1/0 Cross Connect:
1789
1790       DCS1/1 Ports = Ceil((DS1SAL +  Ceil((DS0SAL+ALinkDS0)/24)) / Prt11Max)
```

```
1791        DCS1/0 Common Equipment =Ceil(DS1SAL +  Ceil((DS0SAL+ALinkDS0)/24)) /DCS10DS1Max
1792      }
1793       DCS10Comm := Ceil((DS1SAL +  Ceil((DS0SAL+ALinkDS0)/24))/DCS10DS1Max); //RFPC0032
1794       DCS11Prt := Ceil((DS1SAL +  Ceil((DS0SAL+ALinkDS0)/24))/Prt11Max); //RFPC0032
1795     end
1796     else begin
1797       DCS10Comm := 0;
1798       DCS11Prt := 0;
1799     end;
1800
1801     DCS31Comm := 0;      {initialize}
1802     Mux31Comm := 0;      {     "      }
1803     if (FacType = 'H') and (dm_ICM.t_nodes.fieldbyName('Class').AsInteger = 5) and
1804        IsColocated(CLliCode,t_tandems) then begin
1805
1806       if DSX1Jump >  DCS31DS1Max                    //RFPC0032   (RFPC0054 change DS1SAL to DSX1Jump)
1807         then DCS31Comm := Ceil(DSX1Jump/DCS31DS1Max)//RFPC0032
1808         else DCS31Comm := Ceil((DSX3Jump-DS3SAL)/DCS31DS3Max);        //RFPC0032
1809     end
1810     else
1811       MUX31Comm := Ceil(DSX1Jump/56);
1812     t_tandems.IndexName:='';
1813
1814     DCS31Prt := 0;
1815     DCS33Prt := 0;
1816     if DCS31Comm > 0 then begin
1817     {  RFPC0032
1818        DCS3/1 Cross Connect:
1819
1820        DCS3/1 Ports = Ceil(DSX1Jump / Prt31Max)
1821        DCS3/3 Ports = Ceil(DSX3Jump / Prt33Max)
1822     }
1823       DCS31Prt := Ceil(DSX1Jump / Prt31Max);  // RFPC0032
1824       DCS33Prt := Ceil((DSX3Jump-DS3SAL) / Prt33Max); // RFPC0032
1825     end;
1826
1827     SDS1Card := 0;
1828     SDS3Card := 0;
1829     if (SONETADM <> 3) or (DS3SAL <> 0) then
```

```
1830      SDS3Card := Ceil(DSX3Jump/DS3PCard)
1831    else
1832      SDS1Card := Ceil(DSX1Jump/DS1PCard);
1833
1834        {  RFPC0032
1835         First define number of DS0-equivalent channels by type of equipment,
1836         according to signal level:
1837
1838         For equipment provisioned at DS3 level, use total equivalent DS0s on
1839         the DSX3 cross-connect:
1840
1841         DSX3DS0 = (24 * SWPORTS) + DS0SAL + ALinkDS0 + (24 * DS1SAL) + (672 * DS3SAL)
1842
1843         For equipment provisioned at DS1 level for switched and non-switched DS0s,
1844         use total equivalent DS0s on the DSX1 cross-connect:
1845
1846         DSX1DS0 = (24 * SWPORTS) + DS0SAL + ALinkDS0 + (24 * DS1SAL)
1847
1848         Use total equivalent DS0s on special access DS0s and DS1s for the DCS1/0
1849         cross-connect:
1850
1851         DCS10DS0 = DS0SAL + ALinkDS0 + (24 * DS1SAL)
1852      }
1853       DSX3DS0  := (24.0 * SWPORTS) + DS0SAL + ALinkDS0 + (24.0 * DS1SAL) + (672.0 * DS3SAL); //RFPC0032
1854       DSX1DS0  := (24.0 * SWPORTS) + DS0SAL + ALinkDS0 + (24.0 * DS1SAL);                    //RFPC0032
1855       DCS10DS0 := DS0SAL + ALinkDS0 + (24.0 * DS1SAL);                                       //RFPC0032
1856
1857    //provide the level 4 diagnostics for calculated equipment variables
1858    WeAreWatching.LogEvent(mFUNC,'CalcCLliInvest','   DS0SAL: '+ FloatToStr(DS0SAL)+'   DS1SAL:
1859  '+FloatToStr(DS1SAL)+'   DS3SAL: '+FloatToStr(DS3SAL));
1860    if FacType = 'H' then
1861      WeAreWatching.LogEvent(mFUNC,'CalcCLliInvest','   Remote Switches: '+ FloatToStr(RemoteSW))
1862    else
1863      WeAreWatching.LogEvent(mFUNC,'CalcCLliInvest','   Fiber Modules: '+ FloatToStr(FDPMod)+'  Fiber Comm:
1864  '+FloatToStr(FDPMod));
1865    WeAreWatching.LogEvent(mFUNC,'CalcCLliInvest','   Switch Ports:  '+FloatToStr(SWPorts));
1866    WeAreWatching.LogEvent(mFUNC,'CalcCLliInvest','   DSX1 Jumpers:  '+FloatToStr(DSX1Jump) +'  DSX1 Common:
1867  '+FloatToStr(DSX1Comm));
```

```
1868    WeAreWatching.LogEvent(mFUNC,'CalcCliInvest','   DSX3 Jumpers:  '+FloatToStr(DSX3Jump) +'  DSX3 Common:
1869  '+FloatToStr(DSX3Comm));
1870    WeAreWatching.LogEvent(mFUNC,'CalcCliInvest','   CB Common:     '+FloatToStr(CBComm)   +'  CBCard:
1871  '+FloatToStr(CBCard));
1872    WeAreWatching.LogEvent(mFUNC,'CalcCliInvest','   DCS10 Common:  '+FloatToStr(DCS10Comm)+'  DCS11 Ports:
1873  '+FloatToStr(DCS11Prt));
1874    WeAreWatching.LogEvent(mFUNC,'CalcCliInvest','   DCS31 Common:  '+FloatToStr(DCS31Comm)+'  DCS31 Ports:
1875  '+FloatToStr(DCS31Prt));
1876    WeAreWatching.LogEvent(mFUNC,'CalcCliInvest','   DCS33 Ports:   '+FloatToStr(DCS33Prt) +'  Sonet ADM:
1877  '+FloatToStr(SonetADM));
1878    WeAreWatching.LogEvent(mFUNC,'CalcCliInvest','   Sonet DS1Card: '+FloatToStr(SDS1Card) +'  Sonet
1879  DS3Card: '+FloatToStr(SDS3Card));
1880
1881    {output pass-thru Interconnect components}
1882    {Electrical Interconnect Jumpers}
1883    IOF_Table.Write(CliCode, EquipActNo, 'DSIEJDS0', 'I',
1884      f_OSPMaterial.Mat('DSIEJ',0),f_OSPMaterial.Mat('DSIEJ',0),0,writeRemotes);
1885    IOF_Table.Write(CliCode, EquipActNo, 'DSIEJDS1', 'I',
1886      f_OSPMaterial.Mat('DSIEJ',1),f_OSPMaterial.Mat('DSIEJ',1),0,writeRemotes);
1887    IOF_Table.Write(CliCode, EquipActNo, 'DSIEJDS3', 'I',
1888      f_OSPMaterial.Mat('DSIEJ',3),f_OSPMaterial.Mat('DSIEJ',3),0,writeRemotes);
1889    {optical interconnect jumeprs}
1890    IOF_Table.Write(CliCode, EquipActNo, 'DSIOJDS3', 'I',
1891      f_OSPMaterial.Mat('DSIOJ',3), f_OSPMaterial.Mat('DSIOJ',3),0,writeRemotes);
1892    {Tie (equipment) cables }
1893    IOF_Table.Write(CliCode, EquipActNo, 'DSITDS0', 'I',
1894      f_OSPMaterial.Mat('DSIET',0),f_OSPMaterial.Mat('DSIET',0),0,writeRemotes);
1895    IOF_Table.Write(CliCode, EquipActNo, 'DSITDS1', 'I',
1896      f_OSPMaterial.Mat('DSIET',1),f_OSPMaterial.Mat('DSIET',1),0,writeRemotes);
1897    IOF_Table.Write(CliCode, EquipActNo, 'DSITDS3', 'I',
1898      f_OSPMaterial.Mat('DSIET',3),f_OSPMaterial.Mat('DSIET',3),0,writeRemotes);
1899    {Interconnect terminal blocks}
1900    IOF_Table.Write(CliCode, EquipActNo, 'DSITBDS0', 'I',
1901      f_OSPMaterial.Mat('DSITB',0),f_OSPMaterial.Mat('DSITB',0),0,writeRemotes);
1902    IOF_Table.Write(CliCode, EquipActNo, 'DSITBDS1', 'I',
1903      f_OSPMaterial.Mat('DSITB',1), f_OSPMaterial.Mat('DSITB',1),0,writeRemotes);
1904    IOF_Table.Write(CliCode, EquipActNo, 'DSITBDS3', 'I',
1905      f_OSPMaterial.Mat('DSITB',3), f_OSPMaterial.Mat('DSITB',3),0,writeRemotes);
1906
```

```
1907  {output pass-thru Interconnect components Placement cost}
1908   { Cable to Interconnect Cage }
1909   IOF_Table.Write(CllicCode, EquipActNo, 'PT001', 'I',
1910     f_OSPLabor.Labor('PT001',Tier_Flag),f_OSPLabor.Labor('PT001',Tier_Flag),0,writeRemotes);
1911   { DS0 Jumper }
1912   IOF_Table.Write(CllicCode, EquipActNo, 'PT002', 'I',
1913     f_OSPLabor.Labor('PT002',Tier_Flag), f_OSPLabor.Labor('PT002',Tier_Flag),0,writeRemotes);
1914   { DS1 Jumper }
1915   IOF_Table.Write(CllicCode, EquipActNo, 'PT003', 'I',
1916     f_OSPLabor.Labor('PT003',Tier_Flag), f_OSPLabor.Labor('PT003',Tier_Flag),0,writeRemotes);
1917   { DS3 Jumper }
1918   IOF_Table.Write(CllicCode, EquipActNo, 'PT004', 'I',
1919     f_OSPLabor.Labor('PT004',Tier_Flag), f_OSPLabor.Labor('PT004',Tier_Flag),0,writeRemotes);
1920   { Connector Block }
1921   IOF_Table.Write(CllicCode, EquipActNo, 'PT005', 'I',
1922     f_OSPLabor.Labor('PT005',Tier_Flag), f_OSPLabor.Labor('PT005',Tier_Flag),0,writeRemotes);
1923
1924   // RFPC 0031  add IHDS0, IRDS0, IRDS1, IHDS3, IRDS3, IHSWPORTS,IRSWPORTS demand unit components
1925   IOF_Table.Write(CllicCode, '000000', bh+'SWPORTS', 'I', SWPorts, SWPorts, SWPorts, writeRemotes);
1926   IOF_Table.Write(CllicCode, '000000', bh+'DS0', 'I', DS0SAL, DS0SAL, DS0SAL, writeRemotes);
1927   IOF_Table.Write(CllicCode, '000000', bh+'DS1', 'I', DS1SAL, DS1SAL, DS1SAL, writeRemotes);
1928   IOF_Table.Write(CllicCode, '000000', bh+'DS3', 'I', DS3SAL, DS3SAL, DS3SAL, writeRemotes);
1929
1930   {get equipment costs and engineering/installation labor investment}
1931   // 2 accounting records per equipment are written
1932   //      The Investment record:  Total Invest, Total Invest,
1933   //      The Units record:       Total units,  VS units, Total Units
1934
1935   {check to see if the swich type exists in the switch table}
1936   SwType:=dm_ICM.t_Nodes.FieldByName('Switch Type').AsString;
1937   if not dm_ICM.t_Switch.FindKey([SwType]) then
1938    MessageDlg('Switch type for '+CllicCode+' not found in Switch table.',mtError,[mbOK],0)
1939   else if dm_ICM.t_Switch.fieldbyName('Host/Rem').AsString <> 'D'  then begin
1940     {  RFPC0032
1941        Volume Sensitive DSX1 Cross-Connect:
1942
1943          IxDSX1 =  (DSX1DS0 / 24) * [((DSX1 Common Equipment Material Cost
1944                    + DSX1 Common Engineering & Installation Cost) /DSX1Max)
1945                    (DSX1 Jumper Material Cost + DSX1 Jumper Engineering & Installation Cost)]
```

```
1946
1947            where DSX1Max = Maximum DS1s per DSX1
1948        )
1949         iDSX1vs := (DSX1DS0 / 24) *
1950    //RFPC0032
1951                    ((f_OSPMaterial.Mat('DSXC', 1) + f_OSPLabor.Labor('LDSX1','Rural Rate')) /DSX1Max +
1952    //RFPC0032
1953                     (f_OSPMaterial.Mat('DSXJ', 1) + f_OSPLabor.Labor('LDSX1J','Rural Rate')));
1954    //RFPC0032
1955
1956        {DSX1 equipment}
1957        if (DSX1Max*DSX1Comm) > 0 then begin
1958          iDSX1 := f_OSPMaterial.Mat('DSXC', 1) * DSX1Comm;
1959          iDSX1 := iDSX1 + f_OSPMaterial.Mat('DSXJ', 1) * DSX1Jump;
1960          LabInvest := f_OSPLabor.Labor('LDSX1','Rural Rate') * DSX1Comm;
1961          LabInvest := LabInvest + f_OSPLabor.Labor('LDSX1J','Rural Rate')*DSX1Jump;
1962
1963          if (iDSX1 + LabInvest) > 0 then begin
1964            IOF_Table.Write(CliiCode, EquipActNo, bh+'DSX1', 'I',
1965              (iDSX1 + LabInvest), iDSX1vs, (iDSX1 + LabInvest)-iDSX1vs,writeRemotes);
1966
1967            IOF_Table.Write(CliiCode, EquipActNo, bh+'DSX1DS1', 'I',
1968                            DSX1Jump, (DSX1Max*DSX1Comm), DSX1Jump,writeRemotes);    //RFPC0032
1969          end;
1970        end;
1971
1972        {  RFPC0032
1973           Volume Sensitive DSX3 Cross-Connect:
1974
1975           IxDSX3 =  (DSX3DS0 / 672) * [((DSX3 Common Equipment Material Cost
1976                      + DSX3 Common Engineering & Installation Cost) /DSX3Max)
1977                      (DSX3 Module Material Cost + DSX3 Module Engineering & Installation Cost)]
1978
1979           where DSX3Max = Maximum DS3s per DSX3
1980        }
1981        iDSX3vs := (DSX3DS0 / 672) *
1982    //RFPC0032
1983                    ((f_OSPMaterial.Mat('DSXC', 3) + f_OSPLabor.Labor('LDSX3','Rural Rate')) /DSX3Max +
1984    //RFPC0032
```

```
1985                  (f_OSPMaterial.Mat('DSXJ', 3) + f_OSPLabor.Labor('LDSX3J','Rural Rate')));
1986  //RFPC0032
1987
1988      {DSX3 equipment}
1989      if (DSX3Max*DSX3Comm) > 0 then begin
1990        iDSX3 := f_OSPMaterial.Mat('DSXC', 3) * DSX3Comm;
1991        iDSX3 := iDSX3 + f_OSPMaterial.Mat('DSXJ', 3) * DSX3Jump;
1992        LabInvest:= f_OSPLabor.Labor('LDSX3','Rural Rate') * DSX3Comm;
1993        LabInvest := LabInvest + f_OSPLabor.Labor('LDSX3J','Rural Rate')*DSX3Jump;
1994
1995        if (iDSX3 + LabInvest) > 0 then begin
1996          IOF_Table.Write(CliiCode, EquipActNo, bh+'DSX3', 'I',
1997            (iDSX3 + LabInvest), iDSX3vs, (iDSX3 + LabInvest)-iDSX3vs,writeRemotes);  //RFPC0032
1998
1999          IOF_Table.Write(CliiCode, EquipActNo, bh+'DSX3DS3', 'I',
2000                          DSX3Jump, (DSX3Max*DSX3Comm), DSX3Jump,writeRemotes); //RFPC0032
2001        end;
2002      end;
2003
2004      iCBvs := (DS0SAL + ALinkDS0) *
2005                         ((f_OSPMaterial.Mat('CB', 1) + f_OSPLabor.Labor('LCB','Rural Rate'))/CBMax +
2006  //RFPC0032
2007                           f_OSPMaterial.Mat('CBCard', 1) + f_OSPLabor.Labor('LCBCard','Rural Rate'));
2008  //RFPC0032
2009
2010      {Channel Bank equipment}
2011      if (CBMax*CBComm) > 0 then begin
2012        iCB := f_OSPMaterial.Mat('CB', 1) * CBComm;
2013        iCB := iCB + f_OSPMaterial.Mat('CBCard', 1) * CBCard;
2014        LabInvest := f_OSPLabor.Labor('LCB','Rural Rate')*CBComm;
2015        LabInvest := LabInvest + f_OSPLabor.Labor('LCBCard','Rural Rate') * CBCard;
2016        if (iCB + LabInvest) > 0 then begin
2017          IOF_Table.Write(CliiCode, EquipActNo, bh+'CB', 'I',
2018            (iCB + LabInvest), iCBvs, (iCB + LabInvest)-iCBvs,writeRemotes);        //RFPC0032
2019          IOF_Table.Write(CliiCode, EquipActNo, bh+'CBDS0', 'I',
2020                          (DS0SAL + ALinkDS0), (CBMax*CBComm), (DS0SAL + ALinkDS0),writeRemotes); //RFPC0032
2021        end;
2022      end;
2023
```

```
2024      {  RFPC0032
2025         Volume Sensitive DCS1/0 Cross-Connects:
2026
2027         IxDCS10 =  (DCS10DS0 / 24) * [((DCS1/0 Common Equipment Material Cost
2028                    + DCS1/0 Common Engineering & Installation Cost) /DCS10DS1Max)
2029                    + (DCS1/1 Port Module Cost / Prt11Max)]
2030
2031         where DCS10DS1Max = Maximum DS1s per DCS1/0 and
2032               Prt11Max = Maximum DS1s per Port Module.
2033
2034      }
2035      iDCS10vs := (DCS10DS0 / 24) * ((f_OSPMaterial.Mat('DCSC', 10)+f_OSPLabor.Labor('LDCS1','Rural Rate'))
2036  //RFPC0032
2037                                      / DCS10DS1Max +        //RFPC0032
2038                                       (f_OSPMaterial.Mat('DCSP', 11)+f_OSPLabor.Labor('LDCS11P','Rural
2039  Rate'))//RFPC0032
2040                                      / Prt11Max);           //RFPC0032
2041
2042      {DCS equipment}
2043      if (DSX1Max*DSX1Comm) > 0 then begin
2044        iDCS10 := f_OSPMaterial.Mat('DCSC', 10) * DCS10Comm;
2045        iDCS10 := iDCS10 + f_OSPMaterial.Mat('DCSP', 11) * DCS11prt;
2046        LabInvest := f_OSPLabor.Labor('LDCS1','Rural Rate')*DCS10Comm;
2047        LabInvest := LabInvest + f_OSPLabor.Labor('LDCS11P','Rural Rate')*DCS11prt;
2048
2049        if (iDCS10 + LabInvest) > 0 then begin
2050          IOF_Table.Write(ClliCode, EquipActNo, bh+'DCS10', 'I',
2051            (iDCS10 + LabInvest), iDCS10vs, (iDCS10 + LabInvest)-iDCS10vs,writeRemotes); //RFPC0032
2052
2053          IOF_Table.Write(ClliCode, EquipActNo, bh+'DCS10DS1', 'I',
2054                          (DSX1Jump-SWPorts), (DCS10DS1Max*DCS10Comm), (DSX1Jump-SWPorts),writeRemotes);
2055  //RFPC0032
2056        end;
2057      end;
2058      {  RFPC0032
2059         Volume Sensitive DCS3/1 Cross-Connect:
2060
2061         IxDCS31 =  (DSX1DS0 / 24) * [((DCS3/1 Common Equipment Material Cost
2062                    + DCS3/1 Common Engineering & Installation Cost) /DCS31DS1Max)
```

```
2063                    + (DCS3/1 Port Module Cost / Prt31Max)
2064                    + (DCS3/3 Port Module Cost / (28 * Prt33Max))]
2065
2066          where DCS31DS1Max = Maximum DS1s per DCS3/1,
2067          Prt31Max = Maximum DS1s per DCS3/1 Port Module,
2068          Prt33Max = Maximum DS3s per DCS3/3 Port Module, and
2069          28 = Number of DS1s per DS3.
2070       )
2071
2072       iDCS31vs := (DSX1DS0 / 24) * ((f_OSPMaterial.Mat('DCSC', 31)          //RFPC0032
2073                   + f_OSPLabor.Labor('LDCS3','Rural Rate')) /DCS31DS1Max    //RFPC0032
2074                   + (f_OSPMaterial.Mat('DCSP', 31)  +
2075                   + f_OSPLabor.Labor('LDCS31P','Rural Rate'))  / Prt31Max  //RFPC0032
2076                   + (f_OSPMaterial.Mat('DCSP', 33) +
2077                   + f_OSPLabor.Labor('LDCS33P','Rural Rate')) / (28 * Prt33Max));     //RFPC0032
2078
2079       if (DSX3Max*DSX3Comm) > 0 then begin
2080         iDCS31 := f_OSPMaterial.Mat('DCSC', 31) * DCS31Comm;
2081         iDCS31 := iDCS31 + f_OSPMaterial.Mat('DCSP', 31) * DCS31prt;
2082         iDCS31 := iDCS31 + f_OSPMaterial.Mat('DCSP', 33) * DCS33prt;
2083         LabInvest := f_OSPLabor.Labor('LDCS3','Rural Rate') * DCS31Comm;
2084         LabInvest := LabInvest + f_OSPLabor.Labor('LDCS31P','Rural Rate')*DCS31prt;
2085         LabInvest := LabInvest + f_OSPLabor.Labor('LDCS33P','Rural Rate')*DCS33prt;
2086
2087         if (iDCS31 + LabInvest)> 0 then begin
2088           IOF_Table.Write(CliiCode, EquipActNo, bh+'DCS31', 'I',
2089             (iDCS31 + LabInvest), iDCS31vs, (iDCS31 + LabInvest)-iDCS31vs,writeRemotes);   //RFPC0032
2090
2091           IOF_Table.Write(CliiCode, EquipActNo, bh+'DCS31DS1', 'I',    //RFPC 0024
2092                           DSX1Jump, (DCS31DS1Max*DCS31Comm), DSX1Jump,writeRemotes);          //RFPC0032
2093         end;
2094       end;
2095
2096       {  RFPC0032
2097          Volume Sensitive DS3/DS1 Multiplexer:
2098
2099          IxMUX31 =  (DSX1DS0 / 24) * ((MUX3/1 Common Equipment Material Cost
2100                     + MUX3/1 Common Engineering & Installation Cost) / 56)
2101
```

```
2102          where 56 = Maximum DS1 jumpers per DS3/DS1 Multiplexer.
2103        }
2104        iMUX31vs := (DSX1DS0 / 24) *     //RFPC0032
2105                    ((f_OSPMaterial.Mat('MUX', 31) + f_OSPLabor.Labor('LMUX3','Rural Rate')) / 56); //RFPC0032
2106
2107        {multiplexers}
2108        if ((DSX3Max * DSX3Comm) > 0) then begin
2109          iMux31 := f_OSPMaterial.Mat('MUX', 31) * MUX31Comm;
2110          LabInvest := f_OSPLabor.Labor('LMUX3','Rural Rate')*Mux31Comm;
2111
2112          if (iMUX31 + LabInvest) > 0 then begin
2113            IOF_Table.Write(CllicCode, EquipActNo, bh+'MUX31', 'I',
2114              (iMUX31 + LabInvest), iMUX31vs, (iMUX31 + LabInvest)-iMUX31vs,writeRemotes); //RFPC0032
2115
2116            IOF_Table.Write(CllicCode, EquipActNo, bh+'MUX31DS1', 'I',  //RFPC 0024
2117                            DSX1Jump, (56*MUX31Comm), DSX1Jump,writeRemotes);          //RFPC0032
2118          end;
2119        end;
2120
2121      {Sonet ADM's and DS cards
2122        RFPC0032
2123        Volume Sensitive Add/Drop Multiplexers:
2124
2125        IxADM = (DSX3DS0 /672) * [((ADM Material Cost + ADM Common Equipment
2126                Engineering & Installation Cost) / ADMSize)
2127                + ((ADM Interface Card Material Cost
2128                + ADM Interface Card Engineering & Installation Cost) / 3)]
2129
2130        where ADMSize = 3 for OC-3, 12 for OC-12, and 48 for OC-48;
2131        3 = Number of DS-3s per DS-3 interface card.
2132        }
2133
2134        if (DSX3Max*DSX3Comm) > 0 then begin
2135          if (ADMOC48Comm > 0) or (ADMOC12Comm > 0) then begin
2136            iADM := (f_OSPMaterial.Mat('ADM', 48)*ADMOC48Comm) +
2137                    (f_OSPMaterial.Mat('ADM', 12)*ADMOC12Comm);
2138            LabInvest := (f_OSPLabor.Labor('LADM48','Rural Rate')*ADMOC48Comm) +
2139                         (f_OSPLabor.Labor('LADM12','Rural Rate')*ADMOC12Comm);
2140            iADM := iADM + f_OSPMaterial.Mat('DS', 3)*SDS3Card;
```

```
2141          LabInvest := LabInvest + f_OSPLabor.Labor('LDS3','Rural Rate')*SDS3Card;
2142          iADM := iADM * HostInvst;
2143          LabInvest := LabInvest * HostInvst;
2144          iADMvs  := (DSX3DS0 /672.0) * ((iADM + LabInvest) /          //RFPC0032
2145                      ((ADMOC48Comm * 48.0) + (ADMOC12Comm*12))) +     //RFPC0032
2146                      (f_OSPMaterial.Mat('DS', 3) +                    //RFPC0032
2147                      f_OSPLabor.Labor('LDS3','Rural Rate')) / 3.0; //RFPC0032
2148        end
2149        else begin
2150          iADM := f_OSPMaterial.Mat('ADM', 3);
2151          iADM := iADM + f_OSPMaterial.Mat('DS', 3)*SDS3Card;
2152          iADM := iADM + f_OSPMaterial.Mat('DS', 1)*SDS1Card;
2153          LabInvest := f_OSPLabor.Labor('LADM3','Rural Rate');
2154          LabInvest := LabInvest + f_OSPLabor.Labor('LDS3','Rural Rate')*SDS3Card;
2155          LabInvest := LabInvest + f_OSPLabor.Labor('LDS1','Rural Rate')*SDS1Card;
2156          iADM := iADM * HostInvst;
2157          LabInvest := LabInvest * HostInvst;
2158          if DS3SAL > 0
2159             then iADMvs  := (DSX3DS0 /672.0) * ((f_OSPMaterial.Mat('ADM', 3) +   //RFPC0032
2160                             f_OSPLabor.Labor('LADM3','Rural Rate')) / 3.0 +      //RFPC0032
2161                             (f_OSPMaterial.Mat('DS', 3) +                        //RFPC0032
2162                             f_OSPLabor.Labor('LDS3','Rural Rate')) / 3.0  +      //RFPC0032
2163                             (f_OSPMaterial.Mat('DS', 1) +                        //RFPC0032
2164                             f_OSPLabor.Labor('LDS1','Rural Rate')) / DS1PCard)   //RFPC0032
2165             else iADMvs  := (DSX3DS0 /672.0) * ((f_OSPMaterial.Mat('ADM', 3) +   //RFPC0032
2166                             f_OSPLabor.Labor('LADM3','Rural Rate')) / 3.0 +      //RFPC0032
2167                             (f_OSPMaterial.Mat('DS', 1) +                        //RFPC0032
2168                             f_OSPLabor.Labor('LDS1','Rural Rate')) / DS1PCard);  //RFPC0032
2169        end;
2170        // write ADM record
2171        if (iADM + LabInvest) > 0 then begin
2172          IOF_Table.Write(ClliCode, EquipActNo, bh+'ADM', 'I',
2173                (iADM + LabInvest), iADMvs, (iADM + LabInvest)-iADMvs,writeRemotes);  //RFPC0032
2174
2175          IOF_Table.Write(ClliCode, EquipActNo, bh+'ADMDS3', 'I',
2176                DSX3Jump, (ADMOC48Comm*48)+(ADMOC12Comm*12)+(ADMOC3Comm*3), DSX3Jump,writeRemotes);
2177  //RFPC0032
2178        end;
2179      end;
```

```
2180
2181      (Fiber distribution equipment
2182
2183       RFPC0032
2184       Volume Sensitive Fiber Distribution Panel:
2185
2186       IxFDP =  (DSX3DS0 / 672) * [(FDP Common Equipment Material Cost
2187                + FDP Common Engineering & Installation Cost + FDP Module Material Cost
2188                + FDP Module Engineering & Installation Cost) / ADMSize]
2189
2190       where ADMSize = 3 for OC-3, 12 for OC-12, and 48 for OC-48
2191      )
2192       iFDPvs := (DSX3DS0 / 672) *                                                    //RFPC0032
2193                 ((f_OSPMaterial.Mat('FDPC', 1) + f_OSPLabor.Labor('LFDP','Rural Rate')) +    //RFPC0032
2194                  (f_OSPMaterial.Mat('FDPM',1) + f_OSPLabor.Labor('LFDM', 'Rural Rate')))     //RFPC0032
2195                 / 48;                                                              //RFPC0032
2196
2197      if (DSX3Max* DSX3Comm) > 0 then begin
2198        iFDP := f_OSPMaterial.Mat('FDPC', 1) * FDPComm;
2199        LabInvest := f_OSPLabor.Labor('LFDP','Rural Rate')*FDPComm;
2200        iFDP := iFDP + f_OSPMaterial.Mat('FDPM', 1) * FDPMod;
2201        LabInvest := LabInvest + f_OSPLabor.Labor('LFDM','Rural Rate')*FDPMod;
2202        HFactor := (DSX3Jump*28)/((DSX3Jump*28)+ISWPorts);
2203        IFactor := ISWPorts/((DSX3Jump*28)+ISWPorts);
2204
2205        if (iFDP + LabInvest) > 0 then begin
2206          IOF_Table.Write(ClliCode, EquipActNo, bh+'FDP', 'I', (iFDP + LabInvest)*HFactor,
2207            iFDPvs*HFactor, ((iFDP + LabInvest)-iFDPvs)*HFactor,writeRemotes); //RFPC0032
2208          if (IFactor > 0) then
2209            IOF_Table.Write(ClliCode, EquipActNo, 'IIFDP', 'I',(iFDP + LabInvest)*IFactor,
2210              iFDPvs*IFactor, ((iFDP + LabInvest)-iFDPvs)*IFactor, false); //RFPC0032
2211
2212          IOF_Table.Write(ClliCode, EquipActNo, bh+'FDPDS3', 'I',
2213                          DSX3Jump, (ADMOC48Comm*48)+(ADMOC12Comm*12)+(ADMOC3Comm*3), DSX3Jump,writeRemotes);
2214  //RFPC0032
2215          if (ISWPorts > 0) then
2216            IOF_Table.Write(ClliCode, EquipActNo, 'IIFDPDS1', 'I',
2217                          ISWPorts, ((IIADMOC48Comm*48)+(IIADMOC12Comm*12))*28, ISWPorts, false);  //RFPC0032
2218        end;
```

```
2219      end;
2220
2221     end //test for switch type 'REMX'
2222    else WeAreWatching.LogEvent(mNORMAL,'CalcCliInvest','  '+CliCode+' : '+SwType+'  not processed.');
2223
2224    Application.ProcessMessages;
2225  end; {end function Tf_InterOfc.GetCliInvest}
2226
2227  procedure Tf_InterOfc.Help1Click(Sender: TObject);
2228  begin
2229  end;
2230
2231  {-------------------------------------------------------------------------
2232  Respond to Button Click: Process the file
2233  -------------------------------------------------------------------------}
2234  procedure Tf_InterOfc.btnProcessClick(Sender: TObject);
2235  begin
2236    f_associate.SetFileName(100, 'YES');    {demand on}
2237    Process;
2238  end;
2239
2240  {-------------------------------------------------------------------------
2241  change the sort field
2242  -------------------------------------------------------------------------}
2243  procedure Tf_InterOfc.cbSortChange(Sender: TObject);
2244  begin
2245    if TabSet.TabIndex = 0 then begin
2246      if cbSort.ItemIndex = 0 then
2247        dm_ICM.t_Hst2Hst.IndexName := ''
2248      else if cbSort.ItemIndex = 1 then
2249        dm_ICM.t_Hst2Hst.IndexName := 'ByHubCLLI'
2250
2251    end;
2252    if TabSet.TabIndex = 1 then begin
2253      if cbSort.ItemIndex = 0 then
2254        dm_ICM.t_Hst2Rem.IndexName := ''
2255      else if cbSort.ItemIndex = 1 then
2256        dm_ICM.t_Hst2Rem.IndexName := 'HCLLI'
2257      else if cbSort.ItemIndex = 2 then
```

```
2258        dm_ICM.t_Hst2Rem.IndexName := 'RCLLI'
2259      end;
2260    end;
2261
2262    {-------------------------------------------------------------------------------
2263    Cancel processing
2264    -------------------------------------------------------------------------------}
2265    procedure Tf_InterOfc.btnCancelClick(Sender: TObject);
2266    begin
2267      CancelProcess:= True;
2268      pnlReport.Visible := False;
2269      pnlControl.Visible := True;
2270      pnlEntry.Visible := True;
2271      f_InterOfc.Height := 451;
2272    end;
2273
2274    {-------------------------------------------------------------------------------
2275    print the report
2276    -------------------------------------------------------------------------------}
2277    procedure Tf_InterOfc.PrintIOF( Open : Boolean);
2278    begin
2279     end;
2280
2281    {-------------------------------------------------------------------------------
2282    File/Print the contents of the active notebook page's table
2283    -------------------------------------------------------------------------------}
2284    procedure Tf_InterOfc.Print1Click(Sender: TObject);
2285    begin
2286      {print table on active tabset page)
2287      if PrintDialog1.Execute then begin
2288        if TabSet.TabIndex = 0 then begin
2289          GenericPrintTable(dm_ICM.t_Hst2Hst, Nil, 3007, False, 'Host to Host');
2290          WeAreWatching.LogEvent(mNORMAL,'Print1Click','Host to Host Table ['+dm_ICM.t_Hst2Hst.TableName+']
2291    printed.');
2292        end
2293        else if TabSet.TabIndex = 1 then begin
2294          GenericPrintTable(dm_ICM.t_Hst2Rem, Nil, 3008, False,'Host to Remote');
2295          WeAreWatching.LogEvent(mNORMAL,'Print1Click','Host to Remote Table ['+dm_ICM.t_Hst2Rem.TableName+']
2296    printed.');
```

```
2297      end
2298      else if TabSet.TabIndex = 2 then begin
2299        GenericPrintTable(dm_ICM.t_Tandems, Nil, 3011, False,'Tandems');
2300        WeAreWatching.LogEvent(mNORMAL,'Print1Click','Tandems Table ['+dm_ICM.t_Tandems.TableName+']
2301  printed.');
2302      end
2303      else if TabSet.TabIndex = 3 then begin
2304        GenericPrintTable(dm_ICM.t_InterRing, Nil, 3012, False,'InterRing');
2305        WeAreWatching.LogEvent(mNORMAL,'Print1Click','InterRing Table ['+dm_ICM.t_InterRing.TableName+']
2306  printed.');
2307      end;
2308   end;
2309
2310   PrintIOF(False);
2311  end;
2312
2313  {-------------------------------------------------------------------------
2314  Set constants required for calc's
2315  -------------------------------------------------------------------------}
2316  procedure Tf_InterOfc.SetGridConstants;
2317  begin
2318    { Set Run time Options for Outside plant run }
2319    With t_Grid do begin
2320      if Active then Close;
2321      DatabaseName := 'modata';
2322      TableName := 'roosp.db';
2323      TableType := ttDefault;
2324      Exclusive := True;
2325      Open;
2326
2327      Asmp.Engineering_Feeder  := FieldByName('ED Feeder').AsFloat;
2328      Asmp.Engineering_Dist    := FieldByName('ED Distribution').AsFloat;
2329      Asmp.ADMNFILL            := FieldByName('ED Adm Fill').AsFloat;
2330      Asmp.Sharing_F_frgn_Poles:= FieldByName('SFFPoles').AsFloat;
2331      Asmp.Sharing_F_Aerial    := FieldByName('SFA').AsFloat;
2332      Asmp.Sharing_F_Buried    := FieldByName('SFB').AsFloat;
2333      Asmp.Sharing_F_Cond      := FieldByName('SFC').AsFloat;
2334      Asmp.Sharing_D_frgn_Poles:= FieldByName('SDFPoles').AsFloat;
2335      Asmp.Sharing_D_Aerial    := FieldByName('SDA').AsFloat;
```

```
2336      Asmp.Sharing_D_Buried    := FieldByName('SDB').AsFloat;
2337      Asmp.Sharing_D_Cond      := FieldByName('SDC').AsFloat;
2338      Asmp.Dropwire_Size       := FieldByName('DPWire').AsInteger;
2339      Asmp.OSP_Growth          := FieldByName('Growth').AsFloat;
2340      Asmp.OSP_Horizon         := FieldByName('Horizon').AsFloat;
2341      Asmp.ManHole_Spacing     := FieldByName('ManHole Spacing').AsFloat;
2342      Asmp.PullBox_Spacing     := FieldByName('Pull Box Spacing').AsFloat;
2343      Asmp.Pole_Spacing        := FieldByName('Pole Spacing').AsFloat;
2344      Asmp.XCONN_Factor        := FieldByName('XCONN').AsFloat;
2345      Asmp.Percent_GuyWire     := FieldByName('Per GuyWire').AsFloat;
2346      Asmp.Percent_Concrete    := FieldByName('Per Concrete').AsFloat;
2347      Asmp.Percent_Hand        := FieldByName('Per Hand').AsFloat;
2348      Asmp.PoleUsers           := FieldByName('SPU').AsFloat;
2349      Asmp.TrenchUsers         := FieldByName('STU').AsFloat;
2350      Asmp.ConduitUsers        := FieldByName('SCU').AsFloat;
2351      Asmp.UserDistFill        := FieldByName('Fill Dist').AsFloat;
2352      Asmp.UserFeedFill        := FieldByName('Fill Feeder').AsFloat;
2353      Asmp.UserFill            := FieldByName('UseFillFactor').AsBoolean;
2354      Asmp.CU_Loop_Length      := FieldByName('LoopLength').AsInteger;
2355      Asmp.Aer_CU_Span         := FieldByName('AerCuSpan').AsInteger;
2356      Asmp.Bur_CU_Span         := FieldByName('BurCuSpan').AsInteger;
2357      Asmp.Aer_FI_Span         := FieldByName('AerFiSpan').AsInteger;
2358      Asmp.Bur_FI_Span         := FieldByName('BurFiSpan').AsInteger;
2359      Asmp.LnFtManhole         := FieldByName('LnFtManhole').AsFloat;
2360      Asmp.LnFtPullBox         := FieldByName('LnFtPullBox').AsFloat;
2361      Asmp.WellPointDays       := FieldByName('WellPointDays').AsFloat;
2362      Asmp.Percent_Bore        := FieldByName('PerBoring').AsFloat;
2363      Asmp.MinXSize1           := FieldByName('XCSize1').AsFloat;
2364      Asmp.MinXSize2           := FieldByName('XCSize2').AsFloat;
2365
2366      Close;
2367      WeAreWatching.LogEvent(mNORMAL,'SetGridConstants',' Runtime option constants set.');
2368  {
2369      Foreign_Poles  := Asmp.Sharing_F_frgn_Poles; //RFPC 0010
2370      Sharing_Aerial := Asmp.Sharing_F_Aerial;     //RFPC 0010
2371      Sharing_Buried := Asmp.Sharing_F_Buried;     //RFPC 0010
2372      Sharing_UndGnd := Asmp.Sharing_F_Cond;       //RFPC 0010
2373  }
2374      FiberCable     := True;                  //RFPC 0009
```

```
2375    end;
2376  end;
2377
2378  {-------------------------------------------------------------------------
2379  Calculate straight line distance between two sets of lat/long coordinates
2380  -------------------------------------------------------------------------}
2381  function Tf_InterOfc.CalcDistance(CLat, CLong, HLat, HLong:Real): Integer;
2382  var dresult: double;
2383  begin
2384    CLat  := CLat  * 0.01745;
2385    CLong := CLong * 0.01745;
2386    HLat  := HLat  * 0.01745;
2387    HLong := HLong * 0.01745;
2388    dResult := 20891197 * ArcCos((Cos(CLat)*Cos(HLat)*Abs(Cos(CLong-HLong)))+
2389                         (Sin(CLat)*Sin(HLat)));
2390    Result := Trunc(dResult);
2391  end;
2392
2393  {-------------------------------------------------------------------------
2394  Return  ArcCosine of a number ( in radians )
2395  -------------------------------------------------------------------------}
2396  function Tf_InterOfc.ArcCos(x: double): double;
2397  var y: double;
2398  begin
2399    result := 0;
2400    if (x > 1.0) or (x < -1.0) then
2401      exit;  {overflow condition}
2402    if x= 0.0 then result := 1.5707963267948966192313 22
2403    else if x = 1.0 then result := 0
2404    else if x = -1.0 then result := 3.14159
2405    else begin
2406      y := arctan(sqrt(1.0 - Sqr(x))/x);
2407      if x > 0.0 then result := y
2408      else result := y + 3.14159;
2409    end;
2410  end;
2411
2412  {-------------------------------------------------------------------------
2413  Resize the form
```

```
2414  ------------------------------------------------------------------------------}
2415  procedure Tf_InterOfc.FormResize(Sender: TObject);
2416  begin
2417    if Mode = 1 then begin
2418       if (ClientWidth-pnlControl.Width) < 10 then
2419          ClientWidth := pnlControl.Width + 10;
2420          ClientHeight := 451;
2421    end;
2422
2423    pnlEntry.Top    := l_Copyright.Height + 5;
2424    pnlEntry.Left   := 5;
2425    pnlEntry.width  := ClientWidth - 10;
2426    pnlEntry.Height:= ClientHeight - (pnlEntry.Top + pnlControl.Height + Tabset.Height + 10);
2427    TabSet.Top      := pnlEntry.Height + pnlEntry.Top;
2428    TabSet.Left     := 5;
2429    TabSet.Width    := pnlEntry.Width;
2430    pnlControl.Top := TabSet.Top + TabSet.Height + 5;
2431    pnlControl.Left:= (ClientWidth - pnlControl.width) div 2;
2432    pnlReport.Top   := pnlEntry.Top;
2433    pnlReport.Left  := (ClientWidth - pnlControl.width) div 2;
2434  end;
2435
2436  {------------------------------------------------------------------------------
2437  Query the active table
2438  ------------------------------------------------------------------------------}
2439  procedure Tf_InterOfc.Query1Click(Sender: TObject);
2440  begin
2441    WeAreWatching.LogEvent(mNORMAL,'Query1Click','Query Activated.');
2442    if TabSet.TabIndex = 0 then
2443      f_Query.Query(dm_ICM.t_Hst2Hst)
2444    else if TabSet.TabIndex = 1 then
2445      f_Query.Query(dm_ICM.t_Hst2Rem)
2446    else if TabSet.TabIndex = 2 then
2447      f_Query.Query(dm_ICM.t_Tandems)
2448    else if TabSet.TabIndex = 3 then
2449      f_Query.Query(dm_ICM.t_InterRing);
2450  end;
2451
2452  // cancel table build
```

```
2453  procedure Tf_InterOfc.btnBldCancelClick(Sender: TObject);
2454  begin
2455    WeAreWatching.LogEvent(mNORMAL,'btnBldCancelClick',' Build tables cancelled.');
2456    pnlBuild.Visible := False;
2457    pnlControl.Enabled := True;
2458  end;
2459
2460  // ok to build the tables
2461  procedure Tf_InterOfc.btnBldOKClick(Sender: TObject);
2462  begin
2463    WeAreWatching.LogEvent(mNORMAL,'btnBldOkClick',' Build tables activated.');
2464    BuildTables;
2465    pnlBuild.Visible := False;
2466    pnlControl.Enabled := True;
2467    WeAreWatching.LogEvent(mNORMAL,'btnBldOKClick',' Build Complete.');
2468  end;
2469
2470  // select a file via opendialog
2471  function Tf_InterOfc.SelectFile(Title,DefaultExt,Filter,InitialDir: String): String;
2472  begin
2473    od_Iof.Title := Title;
2474    od_Iof.DefaultExt := DefaultExt;
2475    od_Iof.Filter := Filter;
2476    od_Iof.InitialDir := InitialDir;
2477    if od_Iof.Execute
2478      then Result := od_Iof.FileName
2479    else Result := '';
2480  end;
2481
2482  {make sure file exists before proceeding}
2483  procedure Tf_InterOfc.DoesFileExist(Sender: TObject);
2484  begin
2485    if ActiveControl = btnBldCancel then begin
2486      btnBldCancel.Click;
2487      exit;
2488    end;
2489    with sender as TEdit do begin
2490      if not FileExists(Text) then begin
2491        MessageDlg('File not found.', mtError, [mbOk], 0);
```

```
2492        SetFocus;
2493      end
2494      else
2495        exit;
2496    end;
2497  end;
2498
2499  // respond to file|build button click
2500  procedure Tf_InterOfc.BuildClick(Sender: TObject);
2501  begin
2502    WeAreWatching.LogEvent(mNORMAL,'BuildClick',' Show Rebuild Interoffice tables panel.');
2503    pnlBuild.Visible := True;
2504    pnlControl.Enabled := False;
2505    edH2HDB.SetFocus;
2506  end;
2507
2508  // Select Host to host table
2509  procedure Tf_InterOfc.sbH2HClick(Sender: TObject);
2510  begin
2511    edH2HDB.Text := SelectFile('Enter new Host to Host table name',
2512              '*.DB','Database|*.db;*.dbf', AliasPath);
2513    WeAreWatching.LogEvent(mNORMAL,'sbH2HClick','  New Host to Host Tablename: '+ edH2HDB.Text);
2514  end;
2515
2516  // select host to remote table
2517  procedure Tf_InterOfc.sbH2RClick(Sender: TObject);
2518  begin
2519    edH2RDB.Text := SelectFile('Enter new Host to Remote table name',
2520              '*.DB','Database|*.db;*.dbf', AliasPath);
2521    WeAreWatching.LogEvent(mNORMAL,'sbH2HClick','  New Host to Remote Tablename: '+ edH2RDB.Text);
2522  end;
2523
2524  // select unswitched demand text file
2525  procedure Tf_InterOfc.sbUswDClick(Sender: TObject);
2526  begin
2527    edUSDEMD.Text := SelectFile('Enter the IOT demand input file',
2528              '*.CSV','Import Files|*.CSV;*.TXT', AliasPath);
2529    WeAreWatching.LogEvent(mNORMAL,'sbH2HClick','  IOT demand text file: '+ edUSDEMD.Text);
2530  end;
```

```
2531
2532  // select host to host definition text file
2533  procedure Tf_InterOfc.sbH2HDefClick(Sender: TObject);
2534  begin
2535    edH2HCSV.Text := SelectFile('Enter the host to host input file',
2536               '*.CSV','Import Files|*.CSV;*.TXT', AliasPath);
2537    WeAreWatching.LogEvent(mNORMAL,'sbH2HClick','  Host to Host input text file: '+ edH2HCSV.Text);
2538  end;
2539
2540  {-------------------------------------------------------------------------
2541  Host to Remote before post event (t_Hst2Rem): log change
2542  -------------------------------------------------------------------------}
2543  procedure Tf_InterOfc.t_Hst2RemBeforePost(DataSet: TDataSet);
2544  begin
2545    if dsHst2Rem.State in [dsEdit] then
2546      WeAreWatching.LogEvent(mNORMAL,'t_Hst2RemBeforePost','Host/Remote table record changed');
2547  end;
2548
2549  {-------------------------------------------------------------------------
2550  Host to Host before post event (t_Hst2Hst): log change
2551  -------------------------------------------------------------------------}
2552  procedure Tf_InterOfc.t_Hst2HstBeforePost(DataSet: TDataSet);
2553  begin
2554    if dsHst2Hst.State in [dsEdit] then
2555      WeAreWatching.LogEvent(mNORMAL,'t_Hst2HstBeforePost','Host/Host table record changed');
2556  end;
2557
2558  procedure Tf_InterOfc.sbTndDefClick(Sender: TObject);
2559  begin
2560    edTndCSV.Text := SelectFile('Enter the Tandems input file',
2561               '*.CSV','Import Files|*.CSV;*.TXT', AliasPath);
2562    WeAreWatching.LogEvent(mNORMAL,'sbTndDeflick','  Tandems input text file: '+ edTndCSV.Text);
2563  end;
2564
2565  procedure Tf_InterOfc.Common1Click(Sender: TObject);
2566  begin
2567    GoHelp ('MenuIt.hlp', 304);
2568  end;
2569
```

```
2570  procedure Tf_InterOfc.Tab1Click(Sender: TObject);
2571  begin
2572    Case TabSet.TabIndex of
2573     0 : GoHelp ('Datatables.hlp', 414);
2574     1 : GoHelp ('Datatables.hlp', 415);
2575     2 : GoHelp ('Datatables.hlp', 454);
2576     3 : GoHelp ('Datatables.hlp', 453);
2577    end;
2578  end;
2579
2580  //RFPC0059 and RFPC0067  IHRing and IIRing components
2581  procedure Tf_InterOfc.ProcessRingDistance;
2582  var TabName,TClli,pTClli : string;
2583      Ring,pRing,CurrentRec,i : integer;
2584      RingDistance : double;
2585      t_IHRing, t_IIRing : TTable;
2586
2587    //RFPC0059  Compute the IH ring distance for the specified Clli and ring #
2588    function CalcRingDistance(TClli: string; Ring: integer): double;
2589    var Clli : string;
2590        IHRing : double;
2591    begin
2592      With t_IHRing do
2593        begin
2594          SetRange([TClli,Ring],[TClli,Ring]);
2595          First;
2596          IHRing:=0;
2597          while not eof do
2598            begin
2599              if (FieldByName('Distance').AsInteger > 0) then
2600                IHRing:=IHRing+FieldByName('Distance').AsInteger;
2601              Next;
2602            end;
2603        end;
2604      Result:=IHRing/5280;
2605    end; { CalcRingDistance }
2606
2607    // RFPC0067 Compute the II ring distance for the specified Clli and ring #
2608    function CalcIIRingDistance(TClli: string; Ring: integer): double;
```

```
2609    var Clli : string;
2610        IIRing : double;
2611    begin
2612      With t_IIRing do
2613        begin
2614          SetRange([TClli,Ring],[TClli,Ring]);
2615          First;
2616          IIRing:=0;
2617          while not eof do
2618            begin
2619              if (FieldByName('Distance').AsInteger > 0) then
2620                IIRing:=IIRing+FieldByName('Distance').AsInteger;
2621              Next;
2622            end;
2623        end;
2624      Result:=IIRing/5280;
2625    end; { CalcIIRingDistance }
2626
2627    //RFPC0059  output the IHRing component to the ActH2H table
2628    procedure UpdateIOFTable(TClli: string; Ring: integer; RingDistance: double);
2629    var Clli : string;
2630    begin
2631      With dm_ICM.t_Hst2Hst do
2632        begin
2633          SetRange([TClli,Ring],[TClli,Ring]);
2634          First;
2635          while not eof do
2636            begin
2637              Clli:=FieldByName('ECLLI').AsString;
2638              if (Clli=TClli) then
2639                IOF_Table.Write(Clli,'000000','IHRING','I',0,0,0,false)
2640              else IOF_Table.Write(Clli,'000000','IHRING','I',RingDistance,RingDistance,RingDistance,false);
2641              Next;
2642            end;
2643        end;
2644    end; { UpdateIOFTable }
2645
2646    //RFPC0067  Write out the InterRing Ringdistance component to the ActH2H table
2647    procedure UpdateIOFTableII(TClli: string; Ring: integer; RingDistance: double);
```

```
2648    var Clli, ClliSubstitution : string;
2649    begin
2650      // cycle through interring table ranged on tclli and ring number, and write
2651      // out the ring component to the ActH2H table.
2652      With dm_ICM.t_InterRing do
2653        begin
2654          SetRange([TClli,Ring],[TClli,Ring]);
2655          First;
2656          while not eof do
2657            begin
2658              Clli:=FieldByName('ICLLI').AsString;
2659              ClliSubstitution:=FieldByName('ICLLI Substitution').AsString;
2660              if (Clli=TClli) then  // set the ring distance to zero
2661                IOF_Table.Write(ClliSubstitution,'000000','IIRING','I',0,0,0,false)
2662              else
2663  IOF_Table.Write(ClliSubstitution,'000000','IIRING','I',RingDistance,RingDistance,RingDistance,false);
2664              Next;
2665            end;
2666        end;
2667    end; { UpdateIOFTableII }
2668
2669    // RFPC0067  Save out the ring distance in air miles for the ihring and the
2670    // interring.
2671    procedure SaveOutAirRing(ARTableName,ComponentStr: string; IRFlag: boolean);
2672    var t_AirRing: tTable;
2673        Distance: double;
2674        Clli: string;
2675    begin
2676      if IRFlag then   //RFPC0067
2677        begin // require the interring table for a lookup
2678          dm_ICM.t_InterRing.IndexName:='';
2679          if not dm_ICM.t_InterRing.Active then dm_ICM.t_InterRing.Open;
2680        end;
2681
2682      t_AirRing := TTable.Create(nil);
2683      with t_AirRing do begin
2684        if Active then Close;
2685        DatabaseName := 'modata';
2686        TableName := ARTableName;
```

```
2687        TableType := ttParadox;
2688        Exclusive := True;
2689        if Exists then
2690          begin
2691            Open;
2692            while not EOF do
2693              begin
2694                // convert the distance to miles
2695                Distance:=FieldByName('Distance').AsInteger/5280;
2696                Clli:=FieldByName('Clli').AsString;
2697                if IRFlag then
2698                  begin
2699                    // get the clli substitution for the interring record
2700                    if dm_ICM.t_InterRing.FindKey([Clli]) then
2701                      Clli:=dm_ICM.t_InterRing.FieldByName('IClli Substitution').AsString;
2702                  end;
2703                // write out the component
2704                IOF_Table.Write(Clli,'000000',ComponentStr,'I',Distance,Distance,Distance,false);
2705                Next;
2706              end;
2707            Close;
2708          end;
2709      end;
2710      t_AirRing.Free;
2711    end; { SaveOutAirRing }
2712
2713  begin
2714    { Specify the IHRing table }
2715    t_IHRing:=TTable.Create(nil);
2716    with t_IHRing do
2717      begin
2718        DatabaseName:='modata';
2719        TableType := ttParadox;
2720        TableName:='IHRing';
2721        IndexName:='byHubClli';
2722        Open;
2723      end;
2724    { Host to Host Table }
2725    TabName :=  f_associate.GetFileName(3007);
```

```
2726    With dm_ICM.t_Hst2Hst do
2727      begin
2728        if Active then Close;
2729        DatabaseName := ExtractFilePath(TabName);
2730        TableName := ExtractFilename(TabName);
2731        IndexName := 'ByHubClli';
2732        TableType := ttDefault;
2733        Exclusive := False;
2734        Open;
2735
2736        WeAreWatching.LogEvent(mNORMAL,'ProcessRingDistance','');
2737        WeAreWatching.LogEvent(mNORMAL,'ProcessRingDistance',' Processing InterOffice Ring Calculations.');
2738
2739        // init the progress bar
2740        f_Demand.Gauge1.Progress := 0;
2741        f_Demand.l_Progress.Caption := ' Processing Ring Calculations ';
2742        Application.ProcessMessages;
2743
2744        // compute the ring distances and populate the acth2h table with them
2745        pRing:=0;  pTClli:='';  i:=0;
2746        CancelRange;  // cancel all previously set ranges
2747        First;   // go to the top of the table
2748        while not EOF do
2749          begin
2750            i:=i+1;
2751            // update the progress bar
2752            f_Demand.Gauge1.Progress := Round(i / RecordCount * 100.0);
2753            f_Demand.l_Progress.Caption := Format('H2H %d of %d', [i, RecordCount]);
2754            Application.ProcessMessages;
2755
2756            TClli := FieldByName('HubCLLI').AsString;
2757            Ring  := FieldByName('Ring').AsInteger;
2758            CurrentRec := RecNo;   // get the record number for the current record
2759            if (pTClli <> TClli) or (pRing <> Ring) then
2760              begin
2761                if (pTClli <> TClli) then pTClli:=TClli;
2762                if (pRing <> Ring) then pRing:=Ring;
2763                RingDistance:=CalcRingDistance(TClli,Ring);
2764                WeAreWatching.LogEvent(mFUNC,'ProcessRingDistance','  '+TCLLI+
```

```
2765                  ' Ring '+IntToStr(Ring)+' Distance = '+Trim(Format('%12.5f',[RingDistance])));
2766                if (RingDistance > 0) then
2767                  UpdateIOFTable(TClli,Ring,RingDistance);
2768                CancelRange;  // cancel the range and set the record pointer to the original
2769                RecNo:=CurrentRec;  // set the record pointer back to the original
2770              end;
2771            Next;  // go to the next record
2772          end;
2773
2774        IndexName:='';
2775        Close;
2776      end; {with stmt}
2777    {reset the IHRing table and free up the object}
2778    t_IHRing.IndexName:='';
2779    t_IHRing.Close;
2780    t_IHRing.Free;
2781
2782    {save out the interring distances}
2783    t_IIRing:=TTable.Create(nil);
2784    with t_IIRing do
2785      begin
2786        DatabaseName:='modata';
2787        TableType := ttParadox;
2788        TableName:='IIRing';
2789        IndexName:='byHubClli';
2790        Open;
2791      end;
2792    { Cycle through the InterRing Table }
2793    TabName :=  f_associate.GetFileName(3012);
2794    With dm_ICM.t_InterRing do
2795      begin
2796        if Active then Close;
2797        DatabaseName := ExtractFilePath(TabName);
2798        TableName := ExtractFilename(TabName);
2799        IndexName := 'ByHubClli';
2800        TableType := ttDefault;
2801        Exclusive := False;
2802        Open;
2803
```

```
2804        WeAreWatching.LogEvent(mNORMAL,'ProcessRingDistance',' Processing InterRing Calculations.');
2805
2806        // init the progress bar
2807        f_Demand.Gauge1.Progress := 0;
2808        f_Demand.l_Progress.Caption := ' Processing InterRing Calculations ';
2809        Application.ProcessMessages;
2810
2811        // compute the ring distances and populate the acth2h table with them
2812        pRing:=0;  pTClli:='';  i:=0;
2813        CancelRange;
2814        First;
2815        while not EOF do
2816          begin
2817            i:=i+1;
2818            // update the progress bar
2819            f_Demand.Gauge1.Progress := Round(i / RecordCount * 100.0);
2820            f_Demand.l_Progress.Caption := Format('IR %d of %d', [i, RecordCount]);
2821            Application.ProcessMessages;
2822
2823            TClli := FieldByName('iHubCLLI').AsString;
2824            Ring  := FieldByName('iRing').AsInteger;
2825            CurrentRec := RecNo;  // get the current record number
2826            if (TClli <> '') and ((pTClli <> TClli) or (pRing <> Ring)) then
2827              begin
2828                if (pTClli <> TClli) then pTClli:=TClli;
2829                if (pRing <> Ring) then pRing:=Ring;
2830                RingDistance:=CalcIIRingDistance(TClli,Ring);
2831                WeAreWatching.LogEvent(mFUNC,'ProcessRingDistance','  '+TCLLI+
2832                   ' InterRing '+IntToStr(Ring)+' Distance = '+Trim(Format('%12.5f',[RingDistance])));
2833                if (RingDistance > 0) then UpdateIOFTableII(TClli,Ring,RingDistance);
2834                CancelRange;        // cancel the range and reset the record pointer to the original record
2835                RecNo:=CurrentRec;  // set the record pointer to the original record
2836              end;
2837            Next;  // go the next record
2838          end;
2839
2840        IndexName:='';
2841        Close;
2842      end; {with stmt}
```

```
2843    {reset the InterRing Ring table and free up the object}
2844    t_IIRing.IndexName:='';
2845    t_IIRing.Close;
2846    t_IIRing.Free;
2847
2848    {save out the air ring component}
2849    SaveOutAirRing('AirRing','IHAIRRING',false);
2850    {save out the interring air ring component  RFPC0067}
2851    SaveOutAirRing('IIAirRing','IIAIRRING',true);
2852  end;
2853
2854  // select tandem to tandem table
2855  procedure Tf_InterOfc.sbTndClick(Sender: TObject);
2856  begin
2857    edTndDB.Text := SelectFile('Enter new Tandems table name',
2858              '*.DB','Database|*.db;*.dbf', AliasPath);
2859    WeAreWatching.LogEvent(mNORMAL,'sbTndClick','  New Tandems Tablename: '+ edTndDB.Text);
2860  end;
2861
2862  procedure Tf_InterOfc.sbIRClick(Sender: TObject);
2863  begin
2864    edIRDB.Text := SelectFile('Enter new InterRing table name',
2865              '*.DB','Database|*.db;*.dbf', AliasPath);
2866    WeAreWatching.LogEvent(mNORMAL,'sbIRClick','  New InterRing Tablename: '+ edIRDB.Text);
2867  end;
2868
2869  procedure Tf_InterOfc.sbIRCSVClick(Sender: TObject);
2870  begin
2871    edIRCSV.Text := SelectFile('Enter the InterRing input file',
2872              '*.CSV','Import Files|*.CSV;*.TXT', AliasPath);
2873    WeAreWatching.LogEvent(mNORMAL,'sbIRCSVclick','  InterRing input text file: '+ edIRCSV.Text);
2874  end;
2875
2876  end. {end of unit}
2877
```

IOFARC.pas

```
2
3   unit IOFArc;
4
5   interface
6   uses
7     Windows, Messages, SysUtils, Classes, Graphics, Dialogs, StdCtrls, Forms,
8     dbTables, db;
9
10  procedure ProcessIArcFile;    // process the host to host, host to remote connection
11                                // points, update the main ArcFile and create a HRArcFile
12
13  implementation
14
15  uses Demand, ICMDM, Assoc, InterOfc;
16
17  var TFile : TextFile;
18      Air2Route : double;
19
20  {=============================================================================
21  Return  ArcCosine of a number ( in radians )
22
23   Parameters: the number to evaluate.
24  Assumptions: NONE
25      Returns: ArcCosine of number as a double
26  -----------------------------------------------------------------------------}
27  function ArcCos(x: double): double;
28  var y: double;
29  begin
30    result := 0;
31    if (x > 1.0) or (x < -1.0) then
32      exit;  {overflow}
33    if x = 0.0 then result := 1.57079632679489661923132
34    else if x = 1.0 then result := 0
35    else if x = -1.0 then result := 3.14159
36    else begin
```

```
37      y := arctan(sqrt(1.0 - Sqr(x))/x);
38      if x > 0.0 then result := y
39      else result := y + 3.14159;
40    end;
41  end; { ArcCos }
42
43  {=============================================================================
44  Computes the distance, in feet, between a given two points.
45
46   Parameters: CLat type double;  Latitude of start position
47               CLong type double;  Longitude of start position
48               HLat type double;  Latitude of end position
49               HLong type double;  Longitude of end position
50  Assumptions: All coords supplied are in Latitude and Longitude
51      Returns: Distance, in feet, as double
52  -----------------------------------------------------------------------------}
53  function CalcDistance(CLat, CLong, HLat, HLong:double): Double;
54  begin
55    CLat   := CLat  * 0.01745; {convert degrees to radians}
56    CLong  := CLong * 0.01745;
57    HLat   := HLat  * 0.01745;
58    HLong  := HLong * 0.01745;
59    Result := 20891197 * ArcCos((Cos(CLat)*Cos(HLat)*Abs(Cos(CLong-HLong)))+
60                         (Sin(CLat)*Sin(HLat)));
61  end; { CalcDistance }
62
63  procedure GetRouteToWC(t_ArcFile,t_IArcFile,t_HRArcFile: TTable; Clli,RClli: string;
64                         dlcX,dlcY,IOF,HRF,IRF: integer; HRFlag,IRFlag:boolean);
65  var FromX,FromY,ToX,ToY : variant;
66      atWireCenter : boolean;
67      i,X,Y,NumRecs : integer;
68    function FindIndex(X,Y:integer): integer;
69    var Ind: integer;
70        IndexFound : boolean;
71    begin
72      Ind:=1;  IndexFound:=false;
73      while not IndexFound and (Ind <= NumRecs) do
74        if (ToX[Ind]=X) and (ToY[Ind]=Y) then IndexFound:=true
75        else Ind:=Ind+1;
```

```
76        if IndexFound then Result:=Ind
77        else Result:=0;
78      end; {FindIndex}
79    begin
80      Writeln(TFile);
81      Writeln(TFile,'Getting arcs for ',Clli);
82      if (dlcX=0) and (dlcY=0) then
83        begin
84          Writeln(TFile,Clli,' connection is at the wire center.');
85          exit;
86        end;
87
88      // read in all the feeder records
89      with t_ArcFile do
90        begin
91          CancelRange;
92          SetRange([Clli,'1'],[Clli,'1']);
93          First;
94          NumRecs:=RecordCount;
95          // create the arrays required to read in all the records
96          FromX := VarArrayCreate([1,NumRecs], varInteger);
97          FromY := VarArrayCreate([1,NumRecs], varInteger);
98          ToX   := VarArrayCreate([1,NumRecs], varInteger);
99          ToY   := VarArrayCreate([1,NumRecs], varInteger);
100         i:=0;
101         while not eof do
102           begin
103             i:=i+1;
104             FromX[i] := FieldByName('From X').AsInteger;
105             FromY[i] := FieldByName('From Y').AsInteger;
106             ToX[i]   := FieldByName('To X').AsInteger;
107             ToY[i]   := FieldByName('To Y').AsInteger;
108             Next;
109           end;
110         CancelRange;
111       end;
112
113     X:=dlcX;  Y:=dlcY;  atWireCenter:=false;
114     while not atWireCenter do
```

```
115     begin
116       i:=FindIndex(X,Y);
117       if (i > 0) then
118         begin
119           Writeln(TFile,'From  x=',FromX[i],', y=',FromY[i],'  To  x=',
120             ToX[i],', y=',ToY[i]);
121           X:=FromX[i];
122           Y:=FromY[i];
123           // update the IOF and HRF respectively
124           if t_ArcFile.FindKey([Clli,'1',FromX[i],FromY[i],ToX[i],ToY[i]]) then
125             begin
126               // put the record in edit mode
127               t_ArcFile.Edit;
128               // update the IOF only if it contains a value greater than zero
129               if (IOF > 0) then
130                 t_ArcFile.FieldByName('IOF').AsInteger:=t_ArcFile.FieldByName('IOF').AsInteger+IOF;
131               // update the HRF only if it contains a value greater than zero
132               if (HRF > 0) then
133                 t_ArcFile.FieldByName('HRF').AsInteger:=t_ArcFile.FieldByName('HRF').AsInteger+HRF;
134               // update the IRF only if it contains a value greater than zero
135               if (IRF > 0) then
136                 t_ArcFile.FieldByName('IRF').AsInteger:=t_ArcFile.FieldByName('IRF').AsInteger+IRF;
137               // Post the record
138               t_ArcFile.Post;
139               // create a duplicate entry in the HRArcFile containing the same key
140               if HRFlag then
141                 begin
142                   t_HRArcFile.Append;
143                   t_HRArcFile.FieldByName('Clli').AsString:=Clli;
144                   t_HRArcFile.FieldByName('Arc Type').AsString:=t_ArcFile.FieldByName('Arc Type').AsString;
145                   t_HRArcFile.FieldByName('From X').AsInteger:=t_ArcFile.FieldByName('From X').AsInteger;
146                   t_HRArcFile.FieldByName('From Y').AsInteger:=t_ArcFile.FieldByName('From Y').AsInteger;
147                   t_HRArcFile.FieldByName('To X').AsInteger:=t_ArcFile.FieldByName('To X').AsInteger;
148                   t_HRArcFile.FieldByName('To Y').AsInteger:=t_ArcFile.FieldByName('To Y').AsInteger;
149                   t_HRArcFile.FieldByName('Node').AsInteger:=t_ArcFile.FieldByName('Node').AsInteger;
150                   t_HRArcFile.FieldByName('X').AsInteger:=t_ArcFile.FieldByName('X').AsInteger;
151                   t_HRArcFile.FieldByName('Y').AsInteger:=t_ArcFile.FieldByName('Y').AsInteger;
152                   t_HRArcFile.FieldByName('RClli').AsString:=RClli;
153                   t_HRArcFile.FieldByName('Distance').AsInteger:=
```

```
                    Round(CalcDistance(t_IArcFile.FieldByName('Clli A Lat').AsFloat,
                                       t_IArcFile.FieldByName('Clli A Long').AsFloat,
                                       t_IArcFile.FieldByName('Clli B Lat').AsFloat,
                                       t_IArcFile.FieldByName('Clli A Long').AsFloat));
                  t_HRArcFile.FieldByName('Ring
Dist').AsInteger:=t_IArcFile.FieldByName('Distance').AsInteger;
                  t_HRArcFile.Post;
                end;
            end
          else MessageDlg(Clli+' Feeder rec for ('+IntToStr(FromX[i])+','+
                   IntToStr(FromY[i])+',  '+IntToStr(ToX[i])+','+IntToStr(ToY[i])+') not
found.',mtError,[mbOK],0);
          // check if at the wire center
          atWireCenter:=(X=0) and (Y=0);
        end
      else
        begin
          Writeln(TFile,'Unexpected arc termination.');
          exit;
        end;
    end;
end; { GetRouteToWC }

procedure WriteArcRecord(t_ArcFile:TTable; Clli,ArcType:string;
                         FromX,FromY,Node,Distance,DistToOrigin,IOF,HRF,IRF:integer;
                         NodesFlag: boolean);
var PerAerial,PerBuried,PerUndgnd,Bedrock,WaterTable: string;
begin
  with t_ArcFile do
    begin
      CancelRange;
      if FindKey([Clli,ArcType,FromX,FromY,0,0,Node]) then
        begin
          Edit;
          FieldByName('Length').AsInteger:=FieldByName('Length').AsInteger+Distance;
          if (IOF > 0) then
            FieldByName('IOF').AsInteger:=FieldByName('IOF').AsInteger+IOF;
          if (HRF > 0) then
            FieldByName('HRF').AsInteger:=FieldByName('HRF').AsInteger+HRF;
```

```
193             if (IRF > 0) then
194               FieldByName('IRF').AsInteger:=FieldByName('IRF').AsInteger+IRF;
195           end
196         else
197           begin
198             Append;
199             FieldByName('Clli').AsString:=Clli;
200             FieldByName('Arc Type').AsString:=ArcType;
201             FieldByName('From X').AsInteger:=FromX;
202             FieldByName('From Y').AsInteger:=FromY;
203             FieldByName('To X').AsInteger:=0;
204             FieldByName('To Y').AsInteger:=0;
205             FieldByName('Node').AsInteger:=Node;
206             FieldByName('X').AsInteger:=FromX;
207             FieldByName('Y').AsInteger:=FromY;
208             FieldByName('Length').AsInteger:=Distance;
209             FieldByName('DistToOrigin').AsInteger:=DistToOrigin;
210             FieldByName('Demand').AsInteger:=0;
211             FieldByName('IOF').AsInteger:=IOF;
212             FieldByName('HRF').AsInteger:=HRF;
213             FieldByName('IRF').AsInteger:=IRF;
214             if NodesFlag then
215               begin
216                 if DM_ICM.t_Nodes.FindKey([Clli]) then
217                   begin
218                     FieldByName('PerAerial').AsFloat:=0.0;
219                     FieldByName('PerBuried').AsFloat:=1.0;
220                     FieldByName('PerUndgnd').AsFloat:=0.0;
221                     FieldByName('BedRock').AsString:=DM_ICM.t_Nodes.FieldByName('Bedrock').AsString;
222                     FieldByName('WaterTable').AsString:=DM_ICM.t_Nodes.FieldByName('WaterTable').AsString;
223                   end;
224               end;
225           end;
226         Post;
227       end;
228   end; {WriteArcRec}
229
230   function GetDistToOrigin(t_ArcFile: TTable; Clli: string; X,Y: integer): integer;
231   var arcFound : boolean;
```

```
232       DistToOrigin : integer;
233   begin
234     // check if it already at the center
235     if (X=0) and (Y=0) then
236       begin
237         Result:=0;
238         exit;
239       end;
240
241     // cycle through all the feeder recs until the X,Y point is met
242     with t_ArcFile do
243       begin
244         CancelRange;
245         SetRange([Clli,'1'],[Clli,'1']);
246
247         arcFound:=false;  First;
248         while not eof and not arcFound do
249           if (FieldByName('From X').AsInteger=X) and (FieldByName('From Y').AsInteger=Y) then
250             begin
251               arcFound:=true;
252               DistToOrigin:=FieldByName('DistToOrigin').AsInteger;
253             end
254           else if (FieldByName('To X').AsInteger=X) and (FieldByName('To Y').AsInteger=Y) then
255             begin
256               arcFound:=true;
257               DistToOrigin:=FieldByName('Length').AsInteger+FieldByName('DistToOrigin').AsInteger;
258             end
259           else Next;
260         if arcFound then Result:=DistToOrigin
261         else Result:=0;
262       end;
263   end; { GetDistToOrigin }
264
265   procedure UpdateOSConnInfo(Clli: string; OSConn: double);
266   var EClli : string;
267   begin
268     with DM_ICM.t_Hst2Hst do
269       begin
270         // make sure that no ranges have been specified
```

```
271         CancelRange;
272         // go to the top of the table
273         First;
274         while not eof do
275           begin
276             EClli:=FieldByName('EClli').AsString;
277             // write out the record if the GClli is the same
278             if (FieldByName('GClli').AsString = Clli) and (EClli <> Clli) then
279               IOF_Table.Write (EClli, '000000', 'IHOSCONN', 'I', OSConn, OSConn, OSConn, False);
280             Next;
281           end; {while not eof}
282         // put the table record pointer to the top
283         First;
284       end; {with Hst2Hst}
285   end; { UpdateOSConnInfo }
286
287   // RFPC0059 Update the IH ring distance table with the specified info
288   procedure UpdateIHRing(t_IHRing: TTable; Clli,HubClli: string; Ring,Distance: integer);
289   begin
290     if (Distance <= 0) then exit;     // do not process if the distance is zero
291     with t_IHRing do
292       begin
293         if not Active then Open;
294         // sum up the distance if the record exists
295         if FindKey([Clli,HubClli,Ring]) then
296           begin
297             Edit;
298             FieldByName('Distance').AsInteger:=FieldByName('Distance').AsInteger+Distance;
299           end
300         else  // create a new record
301           begin
302             Append;
303             FieldByName('Clli').AsString      := Clli;
304             FieldByName('HubClli').AsString   := HubClli;
305             FieldByName('Ring').AsInteger     := Ring;
306             FieldByName('Distance').AsInteger := Distance;
307           end;
308         Post;  // post the record with all the changes
309       end;
```

```
310  end; {UpdateIHRing}
311
312  // RFPC0067 Update the InterRing ring distance table with the specified info
313  procedure UpdateIIRing(t_IIRing: TTable; Clli,HubClli: string; Ring,Distance: integer);
314  begin
315    if (Distance <= 0) then exit;   // don't process if the distance is zero
316    with t_IIRing do
317      begin
318        if not Active then Open;
319        // sum up the distance if the record already exists
320        if FindKey([Clli,HubClli,Ring]) then
321          begin
322            Edit;
323            FieldByName('Distance').AsInteger:=FieldByName('Distance').AsInteger+Distance;
324          end
325        else  // create a new record
326          begin
327            Append;
328            FieldByName('Clli').AsString      := Clli;
329            FieldByName('HubClli').AsString   := HubClli;
330            FieldByName('Ring').AsInteger     := Ring;
331            FieldByName('Distance').AsInteger := Distance;
332          end;
333        Post;  // post the record with all the changes
334      end;
335  end; {UpdateIIRing}
336
337  procedure ProcessDLCs(t_DLC,t_ArcFile,t_IArcFile,t_HRArcFile,t_IHRing,t_Tandems,t_IIRing: TTable;
338                        ClliA,ClliB,TClli: string; Ring: integer; GClliFlag,IRFlag:boolean);
339  var ClliALat,ClliALong,ClliBLat,ClliBLong,
340      ClliAX,ClliAY,ClliBX,ClliBY,DistMatrix : Variant;
341      NumDLCsA,NumDLCsB,i,j,aInd,bInd,gInd,DistToOrigin : integer;
342      TempDistance,MinDistance,gMinDistance,GClliLat,GClliLong,OSConn : double;
343      IRClliA, IRClliB : string;
344  begin
345    if IRFlag then  //RFPC0067 InterRing flag encountered, use the Substitutions
346      begin
347        // get the colocllis for the interring related cllis
348        IRClliA:=ClliA;
```

```
349         if dm_ICM.t_InterRing.FindKey([ClliA]) and
350            (dm_ICM.t_InterRing.FieldByName('ICLLI Substitution').AsString <> '') then
351           ClliA:=dm_ICM.t_InterRing.FieldByName('ICLLI Substitution').AsString;
352         IRClliB:=ClliB;
353         if dm_ICM.t_InterRing.FindKey([ClliB]) and
354            (dm_ICM.t_InterRing.FieldByName('ICLLI Substitution').AsString <> '') then
355           ClliB:=dm_ICM.t_InterRing.FieldByName('ICLLI Substitution').AsString;
356       end; {IRFlag}
357
358     Append(TFile);
359     Writeln(TFile,'ClliA = ',ClliA,',  ClliB = ',ClliB);
360     // exit the procedure if any of the Clli's are blank
361     if (ClliA = '') or (ClliB = '') then
362       begin
363         ShowMessage('Require both CLLI codes in order to compute the shortest distance.');
364         Writeln(TFile,'Require both CLLI codes in order to compute the shortest distance.');
365         Writeln(TFile);
366         CloseFile(TFile);
367         exit;
368       end;
369
370     with t_DLC do
371       begin
372         // get the number of DLCs for the first Clli
373         CancelRange;
374         SetRange([ClliA],[ClliA]);
375         NumDLCsA:=RecordCount;
376         // get the number of DLCs for the first Clli
377         CancelRange;
378         SetRange([ClliB],[ClliB]);
379         NumDLCsB:=RecordCount;
380
381         Writeln(TFile,'ClliA DLCs = ',NumDLCsA,',  ClliB DLCs = ',NumDLCsB);
382         // exit the routine if no records exist
383         if (NumDLCsA = 0) or (NumDLCsB = 0) then
384           begin
385             ShowMessage('No DLCs exist for one or both CLLIs.');
386             Writeln(TFile,'No DLCs exist for one or both CLLIs.');
387             Writeln(TFile);
```

```
388           CloseFile(TFile);
389           exit;
390         end;
391
392       // go ahead and set up the arrays
393       ClliALat   := VarArrayCreate([1,NumDLCsA], varDouble);
394       ClliALong  := VarArrayCreate([1,NumDLCsA], varDouble);
395       ClliAX     := VarArrayCreate([1,NumDLCsA], varInteger);
396       ClliAY     := VarArrayCreate([1,NumDLCsA], varInteger);
397       ClliBLat   := VarArrayCreate([1,NumDLCsB], varDouble);
398       ClliBLong  := VarArrayCreate([1,NumDLCsB], varDouble);
399       ClliBX     := VarArrayCreate([1,NumDLCsB], varInteger);
400       ClliBY     := VarArrayCreate([1,NumDLCsB], varInteger);
401       DistMatrix := VarArrayCreate([1,NumDLCsA,1,NumDLCsB], varDouble);
402
403       // populate the Lats and Longs for ClliA
404       CancelRange;
405       SetRange([ClliA],[ClliA]);
406       First;
407       i:=0;
408       Writeln(TFile,ClliA,' DLCs');
409       while not eof do
410         begin
411           i:=i+1;
412           ClliALat[i]  := FieldByName('Latitude').AsFloat;
413           ClliALong[i] := FieldByName('Longitude').AsFloat;
414           ClliAX[i]    := FieldByName('X').AsInteger;
415           ClliAY[i]    := FieldByName('Y').AsInteger;
416           Writeln(TFile,'i=',i,',  Lat = ',ClliALat[i],',  Long = ',ClliALong[i],
417                         ',  X = ',ClliAX[i],', Y = ',ClliAY[i]);
418           Next;
419         end;
420       Writeln(TFile);
421       // populate the Lats and Longs for ClliB
422       CancelRange;
423       SetRange([ClliB],[ClliB]);
424       First;
425       i:=0;
426       Writeln(TFile,ClliB,' DLCs');
```

```
427         while not eof do
428           begin
429             i:=i+1;
430             ClliBLat[i]  := FieldByName('Latitude').AsFloat;
431             ClliBLong[i] := FieldByName('Longitude').AsFloat;
432             ClliBX[i]    := FieldByName('X').AsInteger;
433             ClliBY[i]    := FieldByName('Y').AsInteger;
434             Writeln(TFile,'i=',i,',  Lat = ',ClliBLat[i],',  Long = ',ClliBLong[i],
435                           ',  X = ',ClliBX[i],', Y = ',ClliBY[i]);
436             Next;
437           end;
438         Writeln(TFile);
439
440         // populate the distance matrix
441         MinDistance:=1.7E38;  aInd:=0;  bInd:=0;
442         for i:=1 to NumDLCsA do
443           for j:=1 to NUMDLCsB do
444             begin
445               DistMatrix[i,j]:=CalcDistance(ClliALat[i],ClliALong[i],ClliBLat[j],ClliBLong[j]);
446               if (DistMatrix[i,j] < MinDistance) then
447                 begin
448                   MinDistance:=DistMatrix[i,j];
449                   aInd:=i;  bInd:=j;
450                 end;
451             end;
452
453         if GClliFlag then
454           begin
455             // first get the GCLLI lat and long
456             if DM_ICM.t_Hst2Hst.FindKey([ClliA]) then
457               begin
458                 if t_Tandems.FindKey([DM_ICM.t_Hst2Hst.FieldByName('HubCLLI').AsString]) then
459                   begin
460                     GClliLat:=t_Tandems.FieldByName('Lat').AsFloat;
461                     GClliLong:=t_Tandems.FieldByName('Long').AsFloat;
462                     // get the min distance to the GCLLI
463                     gMinDistance:=1.7E38;  gInd:=0;
464                     for i:=1 to NumDLCsA do
465                       begin
```

```
466                       TempDistance:=CalcDistance(ClliALat[i],ClliALong[i],GClliLat,GClliLong);
467                       if (TempDistance < gMinDistance) then
468                         begin
469                           gMinDistance:=TempDistance;
470                           gInd:=i;
471                         end;
472                     end; {for i loop}
473                 end
474               else gInd:=0;
475             end
476           else gInd:=0;
477         end
478       else gInd:=0;
479
480       if (aInd > 0) and (bInd > 0) then
481         begin
482           Writeln(TFile);
483           Writeln(TFile,'Rows = ',NumDLCsA,', Cols = ',NumDLCsB);
484           for i:=1 to NumDLCsA do
485             begin
486               for j:=1 to NUMDLCsB do Write(TFile,Round(DistMatrix[i,j]),' ');
487               Writeln(TFile);
488             end;
489           Writeln(TFile);
490           Writeln(TFile,'Minimum Distance (',aInd,',',bInd,') = ',Round(MinDistance),',  (',
491
492 Trim(format('%12.4f',[Double(ClliALat[aInd])])),',',Trim(format('%12.4f',[Double(ClliALong[aInd])])),') to
493 (',
494
495 Trim(format('%12.4f',[Double(ClliBLat[bInd])])),',',Trim(format('%12.4f',[Double(ClliBLong[bInd])])),')');
496
497           // write out the record in the arc file
498           if (Ring > 0) then
499             begin {Node type 1 and 2; Host to Host}
500               // ClliA has out(2) connection and ClliB has in(1) connection
501               DistToOrigin:=GetDistToOrigin(t_ArcFile,ClliA,ClliAX[aInd],ClliAY[aInd]);
502               if IRFlag then //RFPC0067 Update the arcfile and the ring distance
503                 begin
504                   WriteArcRecord(t_ArcFile,ClliA,'0',ClliAX[aInd],ClliAY[aInd],2,
```

```
505                     Round((MinDistance*Air2Route)/2),DistToOrigin,0,0,4,true);
506                   // update the IIRing table if ClliA is a HubClli
507                   UpdateIIRing(t_IIRing,ClliA,TClli,Ring,DistToOrigin+Round((MinDistance*Air2Route)/2));
508                 end
509               else  // update the arcfile and IH ring distance
510                 begin
511                   WriteArcRecord(t_ArcFile,ClliA,'0',ClliAX[aInd],ClliAY[aInd],2,
512                     Round((MinDistance*Air2Route)/2),DistToOrigin,4,0,0,true);
513                   // update the IHRing table if ClliA is a HubClli  RFPC0059
514                   UpdateIHRing(t_IHRing,ClliA,TClli,Ring,DistToOrigin+Round((MinDistance*Air2Route)/2));
515                 end;
516               // compute the distance to the wire center
517               DistToOrigin:=GetDistToOrigin(t_ArcFile,ClliB,ClliBX[bInd],ClliBY[bInd]);
518               if IRFlag then  //RFPC0067  update the arcfile and the II ring distance
519                 begin
520                   WriteArcRecord(t_ArcFile,ClliB,'0',ClliBX[bInd],ClliBY[bInd],1,
521                       Round((MinDistance*Air2Route)/2),DistToOrigin,0,0,4,true);
522                   // update the IIRing table if ClliB is a HubClli
523                   UpdateIIRing(t_IIRing,ClliB,TClli,Ring,DistToOrigin+Round((MinDistance*Air2Route)/2));
524                 end
525               else  // update the arcfile and IH ring distance
526                 begin
527                   WriteArcRecord(t_ArcFile,ClliB,'0',ClliBX[bInd],ClliBY[bInd],1,
528                       Round((MinDistance*Air2Route)/2),DistToOrigin,4,0,0,true);
529                   // update the IHRing table if ClliB is a HubClli
530                   UpdateIHRing(t_IHRing,ClliB,TClli,Ring,DistToOrigin+Round((MinDistance*Air2Route)/2));
531                 end;
532               // figure out the path that will be shared for @CLLI
533               if IRFlag then
534 GetRouteToWC(t_ArcFile,t_IArcFile,t_HRArcFile,ClliA,'',ClliAX[aInd],ClliAY[aInd],0,0,4,false,IRFlag)
535               else
536 GetRouteToWC(t_ArcFile,t_IArcFile,t_HRArcFile,ClliA,'',ClliAX[aInd],ClliAY[aInd],4,0,0,false,IRFlag);
537               if IRFlag then
538 GetRouteToWC(t_ArcFile,t_IArcFile,t_HRArcFile,ClliB,'',ClliBX[bInd],ClliBY[bInd],0,0,4,false,IRFlag)
539               else
540 GetRouteToWC(t_ArcFile,t_IArcFile,t_HRArcFile,ClliB,'',ClliBX[bInd],ClliBY[bInd],4,0,0,false,IRFlag);
541               // process the GClliFlag
542               if GClliFlag and (gInd > 0) and not IRFlag then
543                 begin
```

```
544                    DistToOrigin:=GetDistToOrigin(t_ArcFile,ClliA,ClliAX[gInd],ClliAY[gInd]);
545                    OSConn:=(DistToOrigin+Round((gMinDistance*Air2Route)/2))/5280;
546                    WriteArcRecord(t_ArcFile,ClliA,'0',ClliAX[gInd],ClliAY[gInd],4,
547                      Round((gMinDistance*Air2Route)/2),DistToOrigin,4,0,0,true);
548                    // figure out the path that will be shared
549
550  GetRouteToWC(t_ArcFile,t_IArcFile,t_HRArcFile,ClliA,'',ClliAX[gInd],ClliAY[gInd],4,0,0,false,IRFlag);
551                    // write out the OSConn info to the ActH2H table
552                    IOF_Table.Write (ClliA, '000000', 'IHOSCONN', 'I', OSConn, OSConn, OSConn, False);
553                    UpdateOSConnInfo(ClliA,OSConn);
554                  end;
555              end
556            else if (Ring = 0) and not IRFlag then
557              begin {Node type 3; Host to Remote}
558                WriteArcRecord(t_ArcFile,ClliA,'0',ClliAX[aInd],ClliAY[aInd],3,
559                  Round(MinDistance*Air2Route),
560                  GetDistToOrigin(t_ArcFile,ClliA,ClliAX[aInd],ClliAY[aInd]),0,4,0,true);
561                // figure out the path that will be shared for @CLLI
562
563  GetRouteToWC(t_ArcFile,t_IArcFile,t_HRArcFile,ClliA,'',ClliAX[aInd],ClliAY[aInd],0,4,0,false,IRFlag);
564
565  GetRouteToWC(t_ArcFile,t_IArcFile,t_HRArcFile,ClliB,ClliA,ClliBX[bInd],ClliBY[bInd],0,4,0,true,IRFlag);
566              end;
567          end;
568
569        Writeln(TFile,'-----------------------------------------------');
570        Writeln(TFile);
571      end;
572    CloseFile(TFile);
573  end; { ProcessDLCs }
574
575  procedure CreateHRArcFile(t_HRArcFile: TTable);
576  begin
577    with t_HRArcFile do
578      begin
579        if Active then  Close;
580        DatabaseName := 'modata';
581        TableName := 'HRArcFile';
582        TableType := ttParadox;
```

```
583         Exclusive := True;
584         with FieldDefs do
585           begin
586             Clear;
587             Add ('CLLI',         ftString,  11, True);
588             Add ('Arc Type',     ftString,   1, True);
589             Add ('From X',      ftInteger,   0, True);
590             Add ('From Y',      ftInteger,   0, True);
591             Add ('To X',        ftInteger,   0, True);
592             Add ('To Y',        ftInteger,   0, True);
593             Add ('RCLLI',        ftString,  11, True);
594             Add ('Node',        ftInteger,   0, True);
595             Add ('X',           ftInteger,   0, True);
596             Add ('Y',           ftInteger,   0, True);
597             Add ('Distance',    ftInteger,   0, False);
598             Add ('Ring Dist',   ftInteger,   0, False);
599           end;
600         with IndexDefs do
601           begin
602             Clear;
603             Add('Main', 'CLLI;Arc Type;From X;From Y;To X;To Y;RCLLI', [ixPrimary, ixUnique]);
604           end;
605         CreateTable;
606       end;
607   end; { CreateHRArcFile }
608
609   procedure ProcessIArcFile;
610   var t_DLC, t_IArcFile, t_ArcFile, t_IOFSet, t_IHRing, t_IIRing : TTable;
611       TClli, ClliA, ClliB : string;
612       Ring : integer;
613       GClliFlag,IRFlag : boolean;
614       HeapSt : THeapStatus;
615       i : integer;
616   begin
617     try
618
619       // init the output file
620       AssignFile(TFile,'DistMatrix.txt');
621       Rewrite(TFile);
```

```
622       CloseFile(TFile);
623
624       // instantiate the DLC table and initialize it
625       t_DLC:=TTable.Create(nil);
626       with t_DLC do
627         begin
628           DatabaseName:='modata';
629           TableType := ttParadox;
630           TableName:='DLC';
631           Open;
632         end;
633       // instantiate the inner-office arc table and initialize it
634       t_IArcFile:=TTable.Create(nil);
635       with t_IArcFile do
636         begin
637           DatabaseName:='modata';
638           TableType := ttParadox;
639           TableName:='IArcFile';
640           Open;
641         end;
642       // instantiate the arc table and initialize it
643       t_ArcFile:=TTable.Create(nil);
644       with t_ArcFile do
645         begin
646           DatabaseName:='modata';
647           TableType := ttParadox;
648           TableName:='ArcFile';
649           Open;
650         end;
651
652       // instantiate the IHRing table
653       t_IHRing:=TTable.Create(nil);
654       with t_IHRing do
655         begin
656           DatabaseName:='modata';
657           TableType := ttParadox;
658           TableName:='IHRing';
659           Exclusive := True;
660           with FieldDefs do
```

```
661             begin
662               Clear;
663               Add ('CLLI',         FTString, 11, True);
664               Add ('HubCLLI',      FTString, 11, True);
665               Add ('Ring',         FTInteger, 0, True);
666               Add ('Distance',     FTInteger, 0, False);
667             end;
668           with IndexDefs do
669             begin
670               Clear;
671               Add('Main', 'CLLI;HubClli;Ring', [ixPrimary, ixUnique]);
672               Add('byHubClli', 'HubClli;Ring', [ixCaseInsensitive]);
673             end;
674           CreateTable;
675           Open;
676         end;
677
678       // instantiate the IIRing table
679       t_IIRing:=TTable.Create(nil);
680       with t_IIRing do
681         begin
682           DatabaseName:='modata';
683           TableType := ttParadox;
684           TableName:='IIRing';
685           Exclusive := True;
686           with FieldDefs do
687             begin
688               Clear;
689               Add ('CLLI',         FTString, 11, True);
690               Add ('HubCLLI',      FTString, 11, True);
691               Add ('Ring',         FTInteger, 0, True);
692               Add ('Distance',     FTInteger, 0, False);
693             end;
694           with IndexDefs do
695             begin
696               Clear;
697               Add('Main', 'CLLI;HubClli;Ring', [ixPrimary, ixUnique]);
698               Add('byHubClli', 'HubClli;Ring', [ixCaseInsensitive]);
699             end;
```

```
700           CreateTable;
701           Open;
702         end;
703
704       // require the tandems table for lookup of the HubCLLI
705       with DM_ICM.t_Tandems do
706         begin
707           Active := False;
708           DatabaseName := ExtractFilePath(f_associate.GetFileName(3011));  // get the file location
709           TableName := ExtractFilename(f_associate.GetFileName(3011));     // get the table filename
710           TableType := ttDefault;
711   //        IndexName:='byColoClli';
712           Exclusive := True;
713           Open;
714         end;
715
716       // require the InterRing table for lookup of the Colo CLLIs
717       with DM_ICM.t_InterRing do
718         begin
719           Active := False;
720           DatabaseName := ExtractFilePath(f_associate.GetFileName(3012));  // get the file location
721           TableName := ExtractFilename(f_associate.GetFileName(3012));     // get the table filename
722           TableType := ttDefault;
723           IndexName:='';
724           Exclusive := True;
725           Open;
726         end;
727
728       // create the hrarc table
729       CreateHRArcFile(DM_ICM.t_HRArcFile);
730       DM_ICM.t_HRArcFile.Open;
731
732       // open the ActH2H table
733       IOF_Table.Open;
734
735       // instantiate the runtime options table and get the air to route ratio
736       t_IOFSet:=TTable.Create(nil);
737       with t_IOFSet do
738         begin
```

```
739           DatabaseName:='modata';
740           TableType := ttParadox;
741           TableName:='IOFSet';
742           Open;
743           Air2Route:=t_IOFSet.FieldByName('Air2Route').AsFloat;
744           Close;
745         end;
746
747       // require the host to host table for lookup of the TCLLI name when a
748       // GClliFlag is encountered
749       with DM_ICM.t_Hst2Hst do
750         begin
751           Active := False;
752           DatabaseName := ExtractFilePath(f_associate.GetFileName(3007));  // get the file location
753           TableName := ExtractFilename(f_associate.GetFileName(3007));     // get the table filename
754           TableType := ttDefault;
755           IndexName:='';
756           Exclusive := True;
757           Open;
758         end;
759
760       // init the progress bar
761       f_Demand.Gauge1.Progress := 0;
762       f_Demand.l_Progress.Caption := 'InterOffice Arcs';
763       Application.ProcessMessages;
764
765       // read in an arcfile record and find the two points that are the closest
766       with t_IArcFile do
767         begin
768           IndexName:='ByRing';
769           // process remotes first
770           CancelRange;
771           SetRange([0],[0]);
772           i:=0;
773           if (RecordCount > 0) then
774             while not eof do
775               begin
776                 i:=i+1;
777                 // update the progress bar
```

```
778               f_Demand.Gauge1.Progress := Round(i / RecordCount * 100.0);
779               f_Demand.l_Progress.Caption := Format('H2R Arcs %d of %d', [i, RecordCount]);
780               Application.ProcessMessages;
781
782               TClli:=FieldByName('TCLLI').AsString;
783               ClliA:=FieldByName('CLLI A').AsString;
784               ClliB:=FieldByName('CLLI B').AsString;
785               Ring:=FieldByName('Ring').AsInteger;
786               GClliFlag:=FieldByName('GClli Flag').AsBoolean;
787               IRFlag:=FieldByName('InterRing Flag').AsBoolean;
788               ProcessDLCs(t_DLC,t_ArcFile,t_IArcFile,DM_ICM.t_HRArcFile,t_IHRing,
789                           DM_ICM.t_Tandems,t_IIRing,ClliA,ClliB,TClli,Ring,GClliFlag,IRFlag);
790               Next;
791             end;
792         // process host to host first
793         CancelRange;
794         SetRange([1],[999]);
795         i:=0;
796         if (RecordCount > 0) then
797           while not eof do
798             begin
799               i:=i+1;
800               // update the progress bar
801               f_Demand.Gauge1.Progress := Round(i / RecordCount * 100.0);
802               f_Demand.l_Progress.Caption := Format('H2H Arcs %d of %d', [i, RecordCount]);
803               Application.ProcessMessages;
804
805               TClli:=FieldByName('TCLLI').AsString;
806               ClliA:=FieldByName('CLLI A').AsString;
807               ClliB:=FieldByName('CLLI B').AsString;
808               Ring:=FieldByName('Ring').AsInteger;
809               GClliFlag:=FieldByName('GClli Flag').AsBoolean;
810               IRFlag:=FieldByName('InterRing Flag').AsBoolean;
811               ProcessDLCs(t_DLC,t_ArcFile,t_IArcFile,DM_Icm.t_HRArcFile,t_IHRing,
812                           DM_ICM.t_Tandems,t_IIRing,ClliA,ClliB,TClli,Ring,GClliFlag,IRFlag);
813               Next;
814             end;
815       end; { with t_IArcFile }
816   finally
```

```
817       t_IArcFile.Close;
818       t_IArcFile.Free;
819       DM_Icm.t_HRArcFile.Close;
820       DM_Icm.t_Hst2Hst.Close;
821       DM_Icm.t_Tandems.IndexName:='';
822       DM_Icm.t_Tandems.Close;
823       DM_Icm.t_InterRing.Close;
824       IOF_Table.t_Dist.Close;
825       t_ArcFile.Close;
826       t_ArcFile.Free;
827       t_DLC.Close;
828       t_DLC.Free;
829       t_IHRing.Close;
830       t_IHRing.Free;
831       t_IIRing.Close;
832       t_IIRing.Free;
833       t_IOFSet.Free;
834     end; {try}
835   end; { ProcessIArcFile }
836
837   end.
838
```

MAINDEMANDMAP.pas

```
1
2
3   unit MainDemandMap;
4
5   interface
6
7   uses
8     Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
9     ExtCtrls, Buttons, StdCtrls, ComCtrls, Db, DBTables, ToolWin, Printers,
10    ImgList;
11
12  type
13
14    Tf_DemandGUI = class(TForm)
15      ilTB1HotImages: TImageList;
16      ilTB1DisabledImages: TImageList;
17      ColorDialog1: TColorDialog;
18      coolbarGUI: TCoolBar;
19      toolbarDemand: TToolBar;
20      pDemandAndDetail: TPanel;
21      pDemand: TPanel;
22      ScrollBox1: TScrollBox;
23      iDemand: TImage;
24      splitterInfo: TSplitter;
25      pInfo: TPanel;
26      tbZoomIn: TToolButton;
27      tbZoomOut: TToolButton;
28      ilTB1Images: TImageList;
29      tbSep1: TToolButton;
30      tbDensityMap: TToolButton;
31      tbClusterMap: TToolButton;
32      tbBlankMap: TToolButton;
33      tbSep2: TToolButton;
34      tbFeeder: TToolButton;
35      tbAspectRatio: TToolButton;
36      tbSep3: TToolButton;
```

```
37      tbPrint: TToolButton;
38      tbExit: TToolButton;
39      tbSep4: TToolButton;
40      tbInfo: TToolButton;
41      ToolBar1: TToolBar;
42      cbCLLIs: TComboBox;
43      ToolButton1: TToolButton;
44      GroupBox1: TGroupBox;
45      labelZoomFactor: TLabel;
46      tbLandscape: TToolButton;
47      PrintDialog1: TPrintDialog;
48      rgLines: TRadioGroup;
49      ilDLC: TImageList;
50      tbDLC: TToolButton;
51      labelClli: TLabel;
52      t_Inventory: TTable;
53      pcInfo: TPageControl;
54      tsFeeder: TTabSheet;
55      lvFeeder: TListView;
56      tsDistribution: TTabSheet;
57      lvDistribution: TListView;
58      tbHelp: TToolButton;
59      ToolButton2: TToolButton;
60      tbPrev: TToolButton;
61      tbNext: TToolButton;
62      procedure FormCreate(Sender: TObject);
63      procedure cbCLLIsChange(Sender: TObject);
64      procedure iDemandMouseMove(Sender: TObject; Shift: TShiftState; X,Y: Integer);
65      procedure iDemandClick(Sender: TObject);
66      procedure tbExitClick(Sender: TObject);
67      procedure tbInfoClick(Sender: TObject);
68      procedure tbAspectRatioClick(Sender: TObject);
69      procedure tbFeederClick(Sender: TObject);
70      procedure tbDensityMapClick(Sender: TObject);
71      procedure tbZoomInClick(Sender: TObject);
72      procedure tbZoomOutClick(Sender: TObject);
73      procedure tbLandscapeClick(Sender: TObject);
74      procedure tbPrintClick(Sender: TObject);
75      procedure tbDLCClick(Sender: TObject);
```

```
76      procedure FormShow(Sender: TObject);
77      procedure tbHelpClick(Sender: TObject);
78      procedure tbPrevClick(Sender: TObject);
79    private
80      { Private declarations }
81      xPPI, yPPI : double;
82      xMin, yMin, xMax, yMax, xRange, yRange : integer;
83      xInches, yInches : double;
84      prevClli,Clli : string;
85      ppDemandPtX, ppDemandPtY : integer;
86      TotalLines, ResLines, BusLines : integer;
87      xLeft, xRight, xTop, xBottom : integer;
88      AspectRatio : double;
89      UseAspectRatio, ShowFeederArcs, isLandscape : boolean;
90      ShowInfoPanel,ShowDLC : boolean;
91      xPos,yPos,infoCluster : integer;
92      dpX,dpY,dpCluster,dpTotLines : integer;
93
94      procedure GetMaxMin(CalcNearest:boolean);
95      function  CalcDemandPtSize(ppiX, ppiY : double): integer;
96      procedure CalcDemandPtSizeWithAspRatio(ppiX, ppiY : double;
97                                             var ppDemandPtX,ppDemandPtY: integer);
98      procedure DrawTemplate(ppiX,ppiY: double; iCanvas: TCanvas;
99                             ppDemandPtX,ppDemandPtY: integer;
100                            var xLeft,xRight,xTop,xBottom: integer);
101     procedure PrintText(TextStr: string; x1,y1,x2,y2 : integer;
102                         Align: word; iCanvas: TCanvas);
103     procedure PrintRectangle(x1,y1,x2,y2,Wdth : integer; iCanvas: TCanvas);
104     procedure DrawDemandMap(iCanvas: TCanvas; ppDemandPtX,ppDemandPtY,xLeft,xTop: integer);
105     procedure DrawClusterMap(iCanvas: TCanvas; ppDemandPtX,ppDemandPtY,xLeft,xTop: integer);
106     procedure DrawLegend(ppiX,ppiY: double; ppDemandPtX,ppDemandPtY: integer; iCanvas: TCanvas);
107     procedure DrawArcs(ppiX,ppiY: double; iCanvas: TCanvas;
108                        xLeft,xRight,xTop,xBottom,ppDemandPtX,ppDemandPtY,penWidth: integer);
109     procedure DrawDLCs(ppiX,ppiY: double; iCanvas: TCanvas;
110                        xLeft,xRight,xTop,xBottom,ppDemandPtX,ppDemandPtY: integer);
111     procedure PrintDLCs(ppiX,ppiY: double; iCanvas: TCanvas;
112                         xLeft,xRight,xTop,xBottom,ppDemandPtX,ppDemandPtY: integer);
113     procedure ReadInAllInfo;
114     procedure GetInventoryData;
```

```
115       function  OnDemandPt(mX,mY: integer): boolean;
116     public
117       { Public declarations }
118       ZoomFactor : integer;
119       MaxZoom    : integer;
120
121       function  RandomColor: TColor;
122       procedure ConvertToUnscaled(X,Y: integer; var xPos,yPos: integer);
123     end;
124
125   var
126     f_DemandGUI: Tf_DemandGUI;
127
128   implementation
129   uses Assoc, ICMDM, Globalda;
130
131   const MaxRanges=8;
132   type
133     // contains all the color properties for the grid
134     qLegendType = record
135                     MinValue,MaxValue : longint;
136                     qColors           : TColor;
137                   end;
138     // Feeder linked list structure
139     ptrFeeder = ^FeederRec;
140     FeederRec = record
141                   xFrom,yFrom,xTo,yTo : integer;
142                   nextFeeder : ptrFeeder;
143                 end;
144     // DLC linked list structure
145     ptrDLC = ^DLCRec;
146     DLCRec = record
147                X,Y,ClusterID : integer;
148                Lat,Long : double;
149                ResLines, BusLines, TotLines : integer;
150                nextDLC : ptrDLC;
151              end;
152     // Demand and cluster id linked list structure
153     ptrData = ^DataRec;
```

```
154     DataRec = record
155                 X,Y,ClusterID : integer;
156                 ResLines, BusLines, TotLines : integer;
157                 nextData : ptrData;
158               end;
159   var
160     Legends : array [1 .. MaxRanges] of qLegendType;
161     TotalRanges : integer = 8;
162     ClusterColors : array [1 .. 200] of TColor;
163     pDLC : ptrDLC;
164     pData : ptrData;
165     pFeeder : ptrFeeder;
166
167   {$R *.DFM}
168
169   function CalcNearestNum(Num, eNum : integer): integer;
170   var tempNum, sign : integer;
171   begin
172     if (Num=0) or (eNum <= 0) then
173       begin
174         Result:=0;
175         exit;
176       end;
177
178     tempNum:=Abs(Num);
179     if (Num < 0) then sign:=-1 else sign:=1;
180
181     if ((tempNum mod eNum) > 0) then
182       Result:=((tempNum div eNum)+1)*eNum*sign
183     else Result:=Num;
184   end; { CalcNearestNum }
185
186   function Tf_DemandGUI.CalcDemandPtSize(ppiX, ppiY : double): integer;
187   var totXPixels, totYPixels : integer;
188       xPixelsPerDemandPt, yPixelsPerDemandPt : integer;
189   begin
190     if (xRange <= 0) or (yRange <= 0) then
191       begin
192         Result:=0;
```

```
193         exit;
194       end;
195
196     totXPixels := trunc((XInches-0.75)*ppiX);
197     totYPixels := trunc((YInches-3.0)*ppiY);
198
199     xPixelsPerDemandPt := totXPixels div xRange;
200     if Odd(xPixelsPerDemandPt) then xPixelsPerDemandPt:=xPixelsPerDemandPt-1;
201
202     yPixelsPerDemandPt := totYPixels div yRange;
203     if Odd(yPixelsPerDemandPt) then yPixelsPerDemandPt:=yPixelsPerDemandPt-1;
204
205     if (yPixelsPerDemandPt < xPixelsPerDemandPt) then
206       Result := yPixelsPerDemandPt
207     else Result := xPixelsPerDemandPt;
208   end; { CalcDemandPtSize }
209
210   procedure Tf_DemandGUI.CalcDemandPtSizeWithAspRatio(ppiX, ppiY : double;
211                                                       var ppDemandPtX,ppDemandPtY: integer);
212   var totXPixels, totYPixels : integer;
213       xPixelsPerDemandPt, yPixelsPerDemandPt : integer;
214   begin
215     if (xRange <= 0) or (yRange <= 0) then
216       begin
217         ppDemandPtX:=0;  ppDemandPtY:=0;
218         exit;
219       end;
220
221     totXPixels := trunc((XInches-0.75)*ppiX);
222     totYPixels := trunc((YInches-3.0)*ppiY);
223
224     yPixelsPerDemandPt := totYPixels div yRange;
225     if Odd(yPixelsPerDemandPt) then yPixelsPerDemandPt:=yPixelsPerDemandPt-1;
226
227     if (trunc(yPixelsPerDemandPt/AspectRatio)*xRange <= totXPixels) then
228       begin
229         xPixelsPerDemandPt := trunc(yPixelsPerDemandPt/AspectRatio);
230         if Odd(xPixelsPerDemandPt) then xPixelsPerDemandPt:=xPixelsPerDemandPt-1;
231       end
```

```
232     else
233       begin
234         xPixelsPerDemandPt := trunc((totXPixels div xRange)/AspectRatio);
235         if Odd(xPixelsPerDemandPt) then xPixelsPerDemandPt:=xPixelsPerDemandPt-1;
236         yPixelsPerDemandPt := trunc(xPixelsPerDemandPt*AspectRatio);
237         if Odd(yPixelsPerDemandPt) then yPixelsPerDemandPt:=yPixelsPerDemandPt-1;
238       end;
239
240     ppDemandPtY:=yPixelsPerDemandPt;
241     ppDemandPtX:=xPixelsPerDemandPt;
242
243   {  MessageDlg(Clli+', Aspect Ratio = '+Format('%8.5f',[AspectRatio])+#10#13+
244                 'After calc'+#10#13+
245                 'ppX = '+IntToStr(ppDemandPtX)+#10#13+
246                 'ppY = '+IntToStr(ppDemandPtY)+#10#13+
247                 'Ratio Y/X = '+Format('%8.5f',[ppDemandPtY/ppDemandPtX])+#10#13,
248                mtInformation,[mbOK],0);}
249   end; { CalcDemandPtSizeWithAspRatio }
250
251   procedure Tf_DemandGUI.GetMaxMin(CalcNearest:boolean);
252   var X,Y : integer;
253   begin
254     if (Clli='') then exit;
255
256     with dm_ICM.t_Nodes do
257       begin
258         if not Active then Open;
259         if FindKey([Clli]) then
260           begin
261             if (FieldByName('Width Demand Pt').AsInteger > 0) and
262                (FieldByName('Height Demand Pt').AsInteger > 0) then
263               AspectRatio:=(FieldByName('Height Demand Pt').AsInteger/
264                             FieldByName('Width Demand Pt').AsInteger)
265             else AspectRatio:=1;
266           end
267         else AspectRatio:=1;
268         Close;
269       end;
270
```

```
271     xMax:=0;  xMin:=0;  yMax:=0;  yMin:=0;
272     ResLines:=0;  BusLines:=0;  TotalLines:=0;
273     with dm_ICM.t_Demand do
274       begin
275         if not Active then Open;
276         CancelRange;
277         SetRange([Clli],[Clli]);
278         First;
279         if (RecordCount > 0) then
280           while not eof do
281             begin
282               // get the number of lines
283               ResLines:=Reslines+trunc(FieldByName('Res Lines').AsFloat);
284               BusLines:=Buslines+trunc(FieldByName('Bus Lines').AsFloat);
285               TotalLines:=ResLines+BusLines;
286               // get the X and Y values
287               X:=FieldByName('X').AsInteger;
288               Y:=FieldByName('Y').AsInteger;
289               if (X < xMin) then xMin:=X
290               else if (X > xMax) then xMax:=X;
291               if (Y < yMin) then yMin:=Y
292               else if (Y > yMax) then yMax:=Y;
293               Next;
294             end;
295         Close;
296       end;
297
298     if CalcNearest then
299       begin
300         xMin:=CalcNearestNum(xMin,5);
301         xMax:=CalcNearestNum(xMax,5);
302         yMin:=CalcNearestNum(yMin,5);
303         yMax:=CalcNearestNum(yMax,5);
304       end;
305
306     xRange:=Abs(xMax-xMin)+1;
307     yRange:=Abs(yMax-yMin)+1;
308
309     if not UseAspectRatio then
```

```
310       begin
311         ppDemandPtX:=CalcDemandPtSize(xPPI,yPPI);
312         ppDemandPtY:=ppDemandPtX;
313       end
314     else CalcDemandPtSizeWithAspRatio(xPPI,yPPI,ppDemandPtX,ppDemandPtY);
315   end;
316
317   function Tf_DemandGUI.RandomColor;
318   var
319     red, green, blue: Byte;
320   begin
321     red := Random(255);
322     green := Random(255);
323     blue := Random(255);
324
325     Result := red or (green shl 8) or (blue shl 16);
326   end;
327
328   procedure Tf_DemandGUI.FormCreate(Sender: TObject);
329   var i,j : integer;
330       TempColor : TColor;
331       ColorFound : boolean;
332       TabName : string;
333   begin
334     // compute the pixels per inch for both axis
335     xPPI := (iDemand.Width/11.0);
336     yPPI := (iDemand.Height/11.0);
337     xInches := 8.5;
338     yInches := 11.0;
339
340     // init the image
341     with iDemand.Canvas do
342       begin
343         Brush.Color:=clLtGray;
344         FillRect(Rect(0,0,iDemand.Width,iDemand.Height));
345       end;
346
347     ZoomFactor:=100;
348     MaxZoom:=400;
```

```
349     labelZoomFactor.Caption:=IntToStr(ZoomFactor)+'%';
350
351     xMax:=0;  xMin:=0;  yMax:=0;  yMin:=0;
352     TotalLines:=0;  ResLines:=0;  BusLines:=0;
353     infoCluster:=-1;
354     pcInfo.ActivePage:=tsFeeder;
355     prevClli:='';  Clli:='';   UseAspectRatio:=false;  AspectRatio:=1;
356     ShowFeederArcs:=false;  isLandscape:=false;
357     ShowInfoPanel:=true;  ShowDLC:=false;
358
359     tbAspectRatio.Down:=UseAspectRatio;
360     tbFeeder.Down:=ShowFeederArcs;
361     tbDensityMap.Enabled:=false;
362     tbClusterMap.Enabled:=true;
363     tbBlankMap.Enabled:=true;
364
365     rgLines.ItemIndex:=0;  // set the default to total lines
366
367     // populate the Clli list
368     cbCLLIs.Items.Clear;
369     TabName:=f_Associate.GetFileName(2003);
370     with dm_ICM.t_Nodes do
371       begin
372         if Active then Close;
373         DatabaseName:=ExtractFilePath(TabName);
374         TableName:=ExtractFileName(TabName);
375         IndexName:='';
376         Open;
377         First;
378         if (RecordCount > 0) then
379           while not eof do
380             begin
381               cbCLLIs.Items.Add(FieldByName('CLLI').AsString);
382               Next;
383             end;
384         Close;
385       end;
386     cbCLLIs.ItemIndex:=-1;
387
```

```
388     for i:=Low(ClusterColors) to High(ClusterColors) do
389       if (i <= TotalRanges) then ClusterColors[i]:=Legends[i].qColors
390       else
391         begin
392           ColorFound:=false;
393           repeat
394             TempColor:=RandomColor;
395             if (i-1 >= Low(ClusterColors)) then
396               for j:=Low(ClusterColors) to i-1 do
397                 if (TempColor=ClusterColors[j]) then
398                   ColorFound:=true;
399           until not ColorFound;
400           ClusterColors[i]:=TempColor;
401         end;
402
403     // init all the pointers
404     pDLC:=nil;   pData:=nil;  pFeeder:=nil;
405
406     // init the list viewers
407     lvFeeder.Items.Clear;
408     lvDistribution.Items.Clear;
409   end;
410
411   procedure Tf_DemandGUI.FormShow(Sender: TObject);
412   var TabName : string;
413   begin
414     // assign the t_Nodes table name, etc.
415     TabName:=f_Associate.GetFileName(2003);
416     with dm_ICM.t_Nodes do
417       begin
418         if Active then Close;
419         DatabaseName:=ExtractFilePath(TabName);
420         TableName:=ExtractFilename(TabName);
421         IndexName:='';
422       end;
423
424     // assign the t_Demand table name, etc.
425     TabName:=f_Associate.GetFileName(3001);
426     with dm_ICM.t_Demand do
```

```
427       begin
428         if Active then Close;
429         DatabaseName:=ExtractFilePath(TabName);
430         TableName:=ExtractFileName(TabName);
431         IndexName:='';
432       end;
433
434     // assign the t_DLC table name, etc.
435     with dm_ICM.t_DLC do
436       begin
437         if Active then Close;
438         DatabaseName:='modata';
439         TableName:='DLC.db';
440         IndexName:='';
441       end;
442
443     // assign the t_Arcs table name, etc.
444     with dm_ICM.t_Arcs do
445       begin
446         if Active then Close;
447         DatabaseName:='modata';
448         TableName:='ArcFile.db';
449         IndexName:='';
450       end;
451
452     // assign the t_Inventory table name, etc.
453     with t_Inventory do
454       begin
455         if Active then Close;
456         if (DatabaseName='') then DatabaseName:='modata';
457         if (TableName='') then TableName:='Inventory.db';
458         IndexName:='';
459       end;
460   end;
461
462   procedure Tf_DemandGUI.cbCLLIsChange(Sender: TObject);
463   begin
464     Clli:=cbCLLIs.Text;
465     if (prevClli <> Clli) then
```

```
466       begin
467         prevClli:=Clli;
468         infoCluster:=-1;
469         ReadInAllInfo;
470         GetMaxMin(true);
471       end;
472     DrawTemplate(xPPI,yPPI,iDemand.Canvas,ppDemandPtX,ppDemandPtY,
473                  xLeft,xRight,xTop,xBottom);
474
475     if not tbDensityMap.Enabled then DrawDemandMap(iDemand.Canvas,ppDemandPtX,ppDemandPtY,xLeft,xTop)
476     else if not tbClusterMap.Enabled then DrawClusterMap(iDemand.Canvas,ppDemandPtX,ppDemandPtY,xLeft,xTop);
477     if not tbDensityMap.Enabled then DrawLegend(xPPI,yPPI,ppDemandPtX,ppDemandPtY,iDemand.Canvas);
478
479     if ShowFeederArcs then
480   DrawArcs(xPPI,yPPI,iDemand.Canvas,xLeft,xRight,xTop,xBottom,ppDemandPtX,ppDemandPtY,2);
481     if ShowDLC then DrawDLCs(xPPI,yPPI,iDemand.Canvas,xLeft,xRight,xTop,xBottom,ppDemandPtX,ppDemandPtY);
482     iDemand.Invalidate;
483   end;
484
485   procedure Tf_DemandGUI.PrintText(TextStr: string; x1,y1,x2,y2 : integer;
486                                    Align: word; iCanvas: TCanvas);
487   Var
488     Rect  : TRect;                    // Rectangle
489     s     : array[0..255] of char;    // character string
490   begin
491     SetRect( Rect, x1, y1, x2, y2);
492     StrPCopy(S, TextStr);
493     SetBkMode(iCanvas.Handle, 1);
494     DrawText(iCanvas.Handle, S, StrLen(S), Rect, Align);
495   end; { PrintText }
496
497   procedure Tf_DemandGUI.PrintRectangle(x1,y1,x2,y2,Wdth : integer; iCanvas: TCanvas);
498   begin
499     iCanvas.Pen.Width:=Wdth;
500     iCanvas.Pen.Color:=clGray;
501     iCanvas.Rectangle(x1-Wdth+2,y1-Wdth+2,x2+Wdth-1,y2+Wdth-1);
502   end; { PrintLine }
503
504   procedure Tf_DemandGUI.DrawTemplate(ppiX,ppiY: double; iCanvas: TCanvas;
```

```
505                                ppDemandPtX,ppDemandPtY: integer;
506                                var xLeft,xRight,xTop,xBottom: integer);
507   var NumLines : integer;
508   begin
509     // Set font and style
510     iCanvas.Font.Name  := 'Arial';
511     iCanvas.Font.Color := clBlack;
512     iCanvas.Font.Size  := 20;
513     iCanvas.Font.Style := [fsBold, fsUnderline];
514
515     // show the page orientation
516     iCanvas.Brush.Color:=clWhite;
517     iCanvas.FillRect(Rect(0,0,trunc(xInches*ppiX),trunc(yInches*ppiY)));
518
519     // Clli header
520     PrintText(Clli,0,0,trunc(XInches*ppiX),trunc(2/3*ppiY),dt_SingleLine+dt_VCenter+dt_Center,iCanvas);
521
522     // Total lines
523     iCanvas.Font.Size  := 16;
524     if (rgLines.ItemIndex = 0) then NumLines:=TotalLines
525     else if (rgLines.ItemIndex = 1) then NumLines:=ResLines
526     else if (rgLines.ItemIndex = 2) then NumLines:=BusLines
527     else NumLines:=0;
528     PrintText(rgLines.Items[rgLines.ItemIndex]+' Lines = '+IntToStr(NumLines),0,trunc(2/3*ppiY),
529       trunc(XInches*ppiX),trunc(ppiY),dt_Center+dt_SingleLine+dt_VCenter,iCanvas);
530
531     // set the x,y range for the demand map
532     xTop:=trunc(1.5*ppiY);
533     xLeft:=(trunc(xInches*ppiX)-(xRange*ppDemandPtX)) div 2;
534     xRight:=xLeft+(xRange*ppDemandPtX);
535     xBottom:=xTop+(yRange*ppDemandPtY);
536   end; { DrawTemplate }
537
538   function Tf_DemandGUI.OnDemandPt(mX,mY: integer): boolean;
539   var xN,yN,NumLines : integer;
540       dpFound : boolean;
541       tData : ptrData;
542   begin
543     // first check if the mouse cursor is within the rectangle
```

```
544     if (mX >= xLeft) and (mX <= xRight) and (mY >= xTop) and (mY <= xBottom) then
545       begin
546         // convert the screen cord to a corresponding X,Y coord
547         xN  := ((mX-xLeft) div ppDemandPtX)+1;
548         yN  := ((mY-xTop) div ppDemandPtY)+1;
549         dpX := xMin+xN-1;
550         dpY := yMax-yN+1;
551         // now check if the demand pt contains demand
552         if (pData = nil) then Result:=false
553         else
554           begin
555             dpFound:=false;  tData:=pData;
556             while not dpFound and (tData <> nil) do
557               if (tData^.X = dpX) and (tData^.Y = dpY) then
558                 dpFound:=true
559               else tData:=tData^.nextData;
560             if dpFound then
561               begin
562                 if (rgLines.ItemIndex = 0) then NumLines:=tData^.totLines
563                 else if (rgLines.ItemIndex = 1) then NumLines:=tData^.ResLines
564                 else if (rgLines.ItemIndex = 2) then NumLines:=tData^.BusLines
565                 else NumLines:=0;
566                 Result:=(not tbDensityMap.Enabled and (NumLines >= 0)) or
567                         (not tbClusterMap.Enabled and (NumLines > 0));
568               end
569             else Result:=false;
570             if Result then
571               begin
572                 dpCluster  :=tData^.ClusterID;
573                 dpTotLines :=tData^.TotLines;
574               end;
575           end;
576       end
577     else result:=false;
578   end; { OnDemandPt }
579
580   procedure Tf_DemandGUI.ConvertToUnscaled(X,Y: integer; var xPos,yPos: integer);
581   var i,Mult : integer;
582       x1,y1 : double;
```

```
583  begin
584    if (ZoomFactor = 100) then
585      begin
586        xPos:=X;
587        yPos:=Y;
588      end
589    else if (ZoomFactor > 100) then
590      begin
591        x1:=X;  y1:=Y;
592        Mult:=(ZoomFactor-100) div 10;
593        for i:=1 to Mult do
594          begin
595            x1:=(x1*(100/110));
596            y1:=(y1*(100/110));
597          end;
598        xPos:=Trunc(x1);
599        yPos:=Trunc(y1);
600      end
601    else if (ZoomFactor < 100) then
602      begin
603        x1:=X;  y1:=Y;
604        Mult:=(100-ZoomFactor) div 10;
605        for i:=1 to Mult do
606          begin
607            x1:=(x1*(110/100));
608            y1:=(y1*(110/100));
609          end;
610        xPos:=Trunc(x1);
611        yPos:=Trunc(y1);
612      end;
613  end; { ConvertToUnscaled }
614
615  procedure Tf_DemandGUI.iDemandMouseMove(Sender: TObject; Shift: TShiftState; X,
616    Y: Integer);
617  begin
618    ConvertToUnscaled(X,Y,xPos,yPos);
619
620    if (yPos > trunc((YInches-1)*yPPI)) and (yPos < trunc((YInches-0.1)*yPPI)) and
621       (xPos > trunc(xPPI*0.25)) and (xPos < trunc((XInches-0.25)*xPPI)) then
```

```
622       iDemand.Cursor:=crHandPoint
623     else if OnDemandPt(xPos,yPos) then iDemand.Cursor:=crHandPoint
624     else iDemand.Cursor:=crCross;
625   end;
626
627   {-------------------------------------------------------------------------
628    Description: GetRangeColor returns the color of the tile on the GSD map
629                 based on the demand and the corresponding legend specs
630                 cord
631      Arguments: Demand - type double;
632        Returns: TColor type
633    Assumptions: the legend specifications have been set
634   -------------------------------------------------------------------------}
635   function GetRangeColor(Demand:double):TColor;
636   var i : integer;
637       found : boolean;
638   begin
639    found:=false;  i:=1;
640    while (i <= TotalRanges) and not found do
641     begin
642      {determine if the demand is in the ith range}
643      if (i < TotalRanges) and (Demand <= Legends[i].MaxValue) then
644       found:=true
645      else if (i=TotalRanges) and (Demand > Legends[i].MinValue) then
646       found:=true
647      else i:=i+1;
648     end;
649    {set the result if found to the ith legend specs}
650    if found then Result:=Legends[i].qColors
651    else Result:=clWhite;
652   end; {GetRangeColor}
653
654   procedure Tf_DemandGUI.DrawDemandMap(iCanvas: TCanvas; ppDemandPtX,ppDemandPtY,xLeft,xTop: integer);
655   var X,Y,xDP,yDP,xN,yN : integer;
656       pDemand : ptrData;
657       NumLines : integer;
658   begin
659     if (pData = nil) then exit;
660
```

```
661     pDemand:=pData;
662     iCanvas.Pen.Width:=1;
663     while (pDemand <> nil) do
664       if (pDemand^.X <> -9999) and (pDemand^.Y <> -9999) then
665         begin
666           // get the X and Y values
667           xDP:=pDemand^.X;
668           yDP:=pDemand^.Y;
669           // compute the number of demand points it is away from the top left corner
670           xN:=xDP-xMin+1;
671           yN:=yMax-yDP+1;
672           // compute the X,Y pixel points to start from
673           X:=xLeft+((xN-1)*ppDemandPtX);
674           Y:=xTop+((yN-1)*ppDemandPtY);
675           // draw the actual point
676           if (rgLines.ItemIndex = 0) then NumLines:=pDemand^.totLines
677           else if (rgLines.ItemIndex = 1) then NumLines:=pDemand^.ResLines
678           else if (rgLines.ItemIndex = 2) then NumLines:=pDemand^.BusLines
679           else NumLines:=-1;
680           if (NumLines >= 0) then
681             begin
682               iCanvas.Brush.Color:=GetRangeColor(NumLines);
683               iCanvas.Pen.Color:=GetRangeColor(NumLines);
684 //              iCanvas.FillRect(Rect(X,Y,X+ppDemandPtX,Y+ppDemandPtY));
685               iCanvas.Ellipse(X+1,Y+1,X+ppDemandPtX-1,Y+ppDemandPtY-1);
686             end;
687           pDemand:=pDemand^.nextData;
688         end
689       else pDemand:=pDemand^.nextData;
690   end; { DrawDemandMap }
691
692   procedure Tf_DemandGUI.DrawClusterMap(iCanvas: TCanvas; ppDemandPtX,ppDemandPtY,xLeft,xTop: integer);
693   var X,Y,xDP,yDP,xN,yN : integer;
694       pDemand : ptrData;
695       NumLines : integer;
696   begin
697     if (pData = nil) then exit;
698
699     pDemand:=pData;
```

```
700     iCanvas.Pen.Width:=1;
701     while (pDemand <> nil) do
702       if (pDemand^.X <> -9999) and (pDemand^.Y <> -9999) then
703         begin
704           // get the X and Y values
705           xDP:=pDemand^.X;
706           yDP:=pDemand^.Y;
707           // compute the number of demand points it is away from the top left corner
708           xN:=xDP-xMin+1;
709           yN:=yMax-yDP+1;
710           // compute the X,Y pixel points to start from
711           X:=xLeft+((xN-1)*ppDemandPtX);
712           Y:=xTop+((yN-1)*ppDemandPtY);
713           // draw the actual point
714           if (rgLines.ItemIndex = 0) then NumLines:=pDemand^.totLines
715           else if (rgLines.ItemIndex = 1) then NumLines:=pDemand^.ResLines
716           else if (rgLines.ItemIndex = 2) then NumLines:=pDemand^.BusLines
717           else NumLines:=-1;
718           if (NumLines > 0) and (pDemand^.ClusterID <> -1) then
719             begin
720               iCanvas.Brush.Color:=ClusterColors[pDemand^.ClusterID];
721               iCanvas.Pen.Color:=ClusterColors[pDemand^.ClusterID];
722 //              iCanvas.FillRect(Rect(X,Y,X+ppDemandPtX,Y+ppDemandPtY));
723               iCanvas.Ellipse(X+1,Y+1,X+ppDemandPtX-1,Y+ppDemandPtY-1);
724             end;
725           pDemand:=pDemand^.nextData;
726         end
727       else pDemand:=pDemand^.nextData;
728   end; { DrawClusterMap }
729
730   procedure Tf_DemandGUI.DrawLegend(ppiX,ppiY: double; ppDemandPtX,ppDemandPtY: integer; iCanvas: TCanvas);
731   var i,j,k,x,y,xGap,yGap,xBoxSize,yBoxSize : integer;
732       TempStr : string;
733   begin
734     iCanvas.Brush.Color:=clWhite;
735     PrintRectangle(trunc(ppiX*0.25),trunc((YInches-1)*ppiY),
736                    trunc((XInches-0.25)*ppiX),trunc((YInches-0.1)*ppiY),
737                    2,iCanvas);
738     xGap:=trunc((XInches-0.5)*ppiX) div 4;
```

```
739     yGap:=trunc(ppiY*0.5);
740     xBoxSize:=trunc(xGap*0.125);
741     yBoxSize:=trunc(yGap*0.5);
742     iCanvas.Pen.Color:=clBlack;
743     iCanvas.Pen.Width:=1;
744     for i:=1 to 8 do
745       begin
746         j:=(i mod 4);
747         if (j = 0) then j:=4;
748         if ((i mod 4)=0) then k:=(i div 4)-1
749         else k:=(i div 4);
750         x:=trunc(ppiX*0.275)+((j-1)*xGap);
751         y:=trunc((YInches-0.9)*ppiY)+(k*yGap);
752         iCanvas.Brush.Color:=Legends[i].qColors;
753         iCanvas.Pen.Color:=Legends[i].qColors;
754         if (ppDemandPtX <= xBoxSize) and (ppDemandPtY <= yBoxSize) then
755           iCanvas.Ellipse(X+(xBoxSize div 2)-(ppDemandPtX div 2)+1,Y+(yBoxSize div 2)-(ppDemandPtY div 2)+1,
756                           X+(xBoxSize div 2)+(ppDemandPtX div 2)-1,Y+(yBoxSize div 2)+(ppDemandPtY div 2)-1)
757         else iCanvas.Ellipse(X,Y,X+xBoxSize,Y+yBoxSize);
758 //        iCanvas.FillRect(Rect(X,Y,X+xBoxSize,Y+yBoxSize));
759         if (i=1) then TempStr:=' 0 '
760         else if (i=MaxRanges) then TempStr:='over '+IntToStr(Legends[i].MinValue)
761         else TempStr:=IntToStr(Legends[i].MinValue)+' - '+IntToStr(Legends[i].MaxValue);
762         iCanvas.Pen.Color:=clBlack;
763         PrintText(TempStr,x+trunc(xBoxSize*1.2),y,x+xGap,y+(trunc(yGap*2) div 4),
764                   dt_SingleLine+dt_VCenter+dt_Left,iCanvas);
765       end;
766   end; { DrawLegend }
767
768   procedure Tf_DemandGUI.DrawArcs(ppiX,ppiY: double; iCanvas: TCanvas;
769                                   xLeft,xRight,xTop,xBottom,ppDemandPtX,ppDemandPtY,penWidth: integer);
770   var xDP,yDP,xDP1,yDP1,x,y,x1,y1,xN,yN,xN1,yN1 : integer;
771       pArc : ptrFeeder;
772   begin
773     if (pFeeder = nil) then exit;
774
775     iCanvas.Pen.Color:=clRed;
776     iCanvas.Pen.Width:=penWidth;
777     pArc:=pFeeder;
```

```
778     while (pArc <> nil) do
779       begin
780         // get the X and Y values
781         xDP  := pArc^.xFrom;
782         yDP  := pArc^.yFrom;
783         xDP1 := pArc^.xTo;
784         yDP1 := pArc^.yTo;
785         // compute the number of demand points it is away from the top left corner
786         xN  := xDP-xMin+1;
787         yN  := yMax-yDP+1;
788         xN1 := xDP1-xMin+1;
789         yN1 := yMax-yDP1+1;
790         // compute the X,Y pixel points to start from
791         X  := xLeft+((xN-1)*ppDemandPtX)+(ppDemandPtX div 2);
792         Y  := xTop+((yN-1)*ppDemandPtY)+(ppDemandPtY div 2);
793         X1 := xLeft+((xN1-1)*ppDemandPtX)+(ppDemandPtX div 2);
794         Y1 := xTop+((yN1-1)*ppDemandPtY)+(ppDemandPtY div 2);
795         iCanvas.MoveTo(X,Y);
796         iCanvas.LineTo(X1,Y1);
797         pArc:=pArc^.nextFeeder;
798       end;
799   end; { DrawArcs }
800
801   procedure Tf_DemandGUI.DrawDLCs(ppiX,ppiY: double; iCanvas: TCanvas;
802                                   xLeft,xRight,xTop,xBottom,ppDemandPtX,ppDemandPtY: integer);
803   var xDP,yDP,x,y,xN,yN : integer;
804       tDLC : ptrDLC;
805       BitMap : TBitMap;
806   begin
807     if (pDLC = nil) then exit;
808
809     iCanvas.Pen.Color:=clRed;
810     tDLC:=pDLC;
811     BitMap:=TBitMap.Create;
812     while (tDLC <> nil) do
813       begin
814         // get the X and Y values
815         xDP  := tDLC^.X;
816         yDP  := tDLC^.Y;
```

```
817         if (xDP=0) and (yDP=0) then ilDLC.GetBitmap(0,BitMap)
818         else ilDLC.GetBitmap(1,BitMap);
819         // compute the number of demand points it is away from the top left corner
820         xN  := xDP-xMin+1;
821         yN  := yMax-yDP+1;
822         // compute the X,Y pixel points to start from
823         X   := xLeft+((xN-1)*ppDemandPtX)+(ppDemandPtX div 2);
824         Y   := xTop+((yN-1)*ppDemandPtY)+(ppDemandPtY div 2);
825         iCanvas.Draw(x-8,y-8,BitMap);
826         tDLC:=tDLC^.nextDLC;
827       end;
828     BitMap.Free;
829   end; { DrawDLCs }
830
831   procedure Tf_DemandGUI.PrintDLCs(ppiX,ppiY: double; iCanvas: TCanvas;
832                                    xLeft,xRight,xTop,xBottom,ppDemandPtX,ppDemandPtY: integer);
833   var xDP,yDP,x,y,xN,yN : integer;
834       tDLC : ptrDLC;
835       TempStr : string;
836   begin
837     if (pDLC = nil) then exit;
838
839     iCanvas.Font.Color := clBlack;
840     iCanvas.Font.Size  := 8;
841     iCanvas.Font.Style := [fsBold];
842     iCanvas.Brush.Color:=clLtGray;
843     tDLC:=pDLC;
844     while (tDLC <> nil) do
845       begin
846         // get the X and Y values
847         xDP  := tDLC^.X;
848         yDP  := tDLC^.Y;
849         if (xDP=0) and (yDP=0) then TempStr:='WC' else TempStr:='DLC';
850         // compute the number of demand points it is away from the top left corner
851         xN  := xDP-xMin+1;
852         yN  := yMax-yDP+1;
853         // compute the X,Y pixel points to start from
854         X   := xLeft+((xN-1)*ppDemandPtX)+(ppDemandPtX div 2);
855         Y   := xTop+((yN-1)*ppDemandPtY)+(ppDemandPtY div 2);
```

```
856         iCanvas.Pen.Color:=clLtGray;
857         iCanvas.FillRect(Rect(x-trunc(ppiX*0.125),y-trunc(ppiY*0.09),
858                               x+trunc(ppiX*0.125),y+trunc(ppiX*0.09)));
859         iCanvas.Pen.Color:=clBlack;
860         PrintText(TempStr,x-trunc(ppiX*0.125),y-trunc(ppiY*0.09),
861                               x+trunc(ppiX*0.125),y+trunc(ppiX*0.09),
862                     dt_SingleLine+dt_VCenter+dt_Center,iCanvas);
863   //      iCanvas.StretchDraw(Rect(x-Round(ppiX*0.09),y-Round(ppiY*0.09),
864   //                               x+Round(ppiX*0.09),y+Round(ppiY*0.09)),BitMap);
865         tDLC:=tDLC^.nextDLC;
866       end;
867   end; { DrawDLCs }
868
869   procedure Tf_DemandGUI.GetInventoryData;
870   var ListItem: TListItem;
871       X,Y : integer;
872       tDLC : ptrDLC;
873       DLCFound : boolean;
874   begin
875     if (dpCluster=infoCluster) then exit;
876
877     // init the list viewers
878     lvFeeder.Items.Clear;
879     lvDistribution.Items.Clear;
880     infoCluster:=dpCluster;
881     labelClli.Caption:=Clli;
882
883      // get the DLC X,Y coords
884      if (pDLC = nil) then exit
885      else
886        begin
887          DLCFound:=false;  tDLC:=pDLC;
888          while not DLCFound and (tDLC <> nil) do
889            if (tDLC^.ClusterID = dpCluster) then
890              DLCFound:=true
891            else tDLC:=tDLC^.nextDLC;
892          if DLCFound then
893            begin
894              X:=tDLC^.X;
```

```
895             Y:=tDLC^.Y;
896           end
897         else exit;
898       end;
899
900   with t_Inventory do
901     begin
902       // open the inventory table
903       IndexName:='ByGSD';
904       if not Active then Open;
905
906       // get all the feeder info
907       CancelRange;
908       SetRange([Clli,'F',X,Y],[Clli,'F',X,Y]);
909       First;
910       if (RecordCount > 0) then
911         while not EOF do
912           begin
913             ListItem := lvFeeder.Items.Add;
914             ListItem.Caption := FieldByName('Component').AsString;
915             ListItem.SubItems.Add(FieldByName('Size').AsString);
916             ListItem.SubItems.Add(FieldByName('Units').AsString);
917             Next;
918           end; {while not eof}
919
920       // get all the distribution info
921       CancelRange;
922       SetRange([Clli,'D',X,Y],[Clli,'D',X,Y]);
923       First;
924       if (RecordCount > 0) then
925         while not EOF do
926           begin
927             ListItem := lvDistribution.Items.Add;
928             ListItem.Caption := FieldByName('Component').AsString;
929             ListItem.SubItems.Add(FieldByName('Size').AsString);
930             ListItem.SubItems.Add(FieldByName('Units').AsString);
931             Next;
932           end; {while not eof}
933       Close;
```

```
934         IndexName:='';
935       end; {with t_Inventory}
936   end; { GetInventoryData }
937
938   procedure Tf_DemandGUI.iDemandClick(Sender: TObject);
939   var i,x,y : integer;
940   begin
941     // Check to see if the legend was clicked
942     if (yPos > trunc((YInches-1)*yPPI)) and (yPos < trunc((YInches-0.1)*yPPI)) and
943        (xPos > trunc(xPPI*0.25)) and (xPos < trunc((XInches-0.25)*xPPI)) then
944       begin
945         y:=((yPos-trunc((YInches-1)*yPPI)) div trunc(yPPI*0.5))+1;
946         x:=((xPos-trunc(xPPI*0.25)) div (trunc((XInches-0.5)*xPPI) div 4))+1;
947         i:=((y-1)*4)+x;
948         ColorDialog1.Color:=Legends[i].qColors;
949         iDemand.Cursor:=crCross;
950         if ColorDialog1.Execute then
951           begin
952             Legends[i].qColors:=ColorDialog1.Color;
953             cbCLLIsChange(Sender);
954           end;
955       end
956     else if (xPos >= xLeft) and (xPos <= xRight) and (yPos >= xTop) and
957             (yPos <= xBottom) and onDemandPt(xPos,yPos) and (dpTotLines > 0) then
958       begin
959         // put in detail for feeder and distribution inventory
960         if (infoCluster <> dpCluster) then GetInventoryData;
961         if not pInfo.Visible then
962           begin
963             pInfo.Visible:=true;
964             splitterInfo.Left:=pDemandAndDetail.Width;
965           end;
966         tbInfo.Down:=False;
967       end;
968   end;
969
970   function NewFeeder: ptrFeeder;
971   var tempFeeder : ptrFeeder;
972   begin
```

```
973     New(tempFeeder);
974     tempFeeder^.xFrom      := -1;
975     tempFeeder^.yFrom      := -1;
976     tempFeeder^.xTo        := -1;
977     tempFeeder^.yTo        := -1;
978     tempFeeder^.nextFeeder := nil;
979
980     Result:=tempFeeder;
981   end; { NewFeeder }
982
983   procedure DisposeFeeder(Feeder : ptrFeeder);
984   begin
985     if (Feeder^.nextFeeder = nil) then exit
986     else DisposeFeeder(Feeder^.nextFeeder);
987     Dispose(Feeder);
988   end; { DisposeFeeder }
989
990   function NewDLC: ptrDLC;
991   var tempDLC : ptrDLC;
992   begin
993     New(tempDLC);
994     tempDLC^.X          := -9999;
995     tempDLC^.Y          := -9999;
996     tempDLC^.Lat        := -9999;
997     tempDLC^.Long       := -9999;
998     tempDLC^.ClusterID  := -1;
999     tempDLC^.ResLines   := 0;
1000    tempDLC^.BusLines   := 0;
1001    tempDLC^.TotLines   := 0;
1002    tempDLC^.nextDLC    := nil;
1003
1004    Result:=tempDLC;
1005  end; { NewDLC }
1006
1007  procedure DisposeDLC(DLC : ptrDLC);
1008  begin
1009    if (DLC^.nextDLC = nil) then exit
1010    else DisposeDLC(DLC^.nextDLC);
1011    Dispose(DLC);
```

```
1012  end; { DisposeDLC }
1013
1014  function NewData: ptrData;
1015  var tempData : ptrData;
1016  begin
1017    New(tempData);
1018    tempData^.X         := -9999;
1019    tempData^.Y         := -9999;
1020    tempData^.BusLines  := 0;
1021    tempData^.ResLines  := 0;
1022    tempData^.TotLines  := 0;
1023    tempData^.ClusterID := -1;
1024    tempData^.nextData  := nil;
1025
1026    Result:=tempData;
1027  end; { NewData }
1028
1029  procedure DisposeData(dData : ptrData);
1030  begin
1031    if (dData^.nextData = nil) then exit
1032    else DisposeData(dData^.nextData);
1033    Dispose(dData);
1034  end; { DisposeData }
1035
1036  function GetClusterID(X,Y: integer): integer;
1037  var TempInt : integer;
1038      ClusterFound : boolean;
1039      TempPtr : ptrDLC;
1040  begin
1041    if (pDLC = nil) then
1042      begin
1043        Result:=-1;
1044        exit;
1045      end;
1046
1047    ClusterFound:=false;  TempPtr:=pDLC;  TempInt:=-1;
1048    while (TempPtr <> nil) and not ClusterFound do
1049      if (TempPtr^.X = X) and (TempPtr^.Y = Y) then
1050        begin
```

```
1051            TempInt:=TempPtr^.ClusterID;
1052            ClusterFound:=true;
1053          end
1054        else TempPtr:=TempPtr^.nextDLC;
1055
1056    Result:=TempInt;
1057  end; { GetClusterID }
1058
1059  function GetDLC(X,Y: integer): ptrDLC;
1060  var TempPtr : ptrDLC;
1061      DLCFound : boolean;
1062  begin
1063    if (pDLC=nil) then
1064      begin
1065        Result:=nil;
1066        exit;
1067      end;
1068
1069    TempPtr:=pDLC;    DLCFound:=false;
1070    while (TempPtr <> nil) and not DLCFound do
1071      if (TempPtr^.X = X) and (TempPtr^.Y = Y) then DLCFound:=true
1072      else TempPtr:=TempPtr^.nextDLC;
1073
1074    if DLCFound then Result:=TempPtr
1075    else Result:=nil;
1076  end; { GetDLC }
1077
1078  procedure AddToClusterLines(ClusterID,ResLines,BusLines,TotLines: integer);
1079  var DLCFound : boolean;
1080      tDLC : ptrDLC;
1081  begin
1082    if (pDLC = nil) then exit;
1083
1084    DLCFound:=false;  tDLC:=pDLC;
1085    while not DLCFound and (tDLC <> nil) do
1086      if (tDLC^.ClusterID = ClusterID) then
1087        DLCFound:=true
1088      else tDLC:=tDLC^.nextDLC;
1089    if DLCFound then
```

```
1090      begin
1091        tDLC^.ResLines:=tDLC^.ResLines+ResLines;
1092        tDLC^.BusLines:=tDLC^.BusLines+BusLines;
1093        tDLC^.TotLines:=tDLC^.TotLines+TotLines;
1094      end;
1095  end; { AddToClusterLines }
1096
1097  procedure Tf_DemandGUI.ReadInAllInfo;
1098  var prevDLC, DLC : ptrDLC;
1099      prevData, dData : ptrData;
1100      prevFeeder, Feeder : ptrFeeder;
1101      ClusterIndex : integer;
1102  begin
1103    // make sure that the previous DLC linked list is disposed before a new
1104    // one is created
1105    if (pDLC <> nil) then
1106      begin
1107        DisposeDLC(pDLC);
1108        pDLC:=nil;
1109      end;
1110
1111    // create the DLC linked list
1112    with dm_ICM.t_DLC do
1113      begin
1114        if not Active then Open;
1115        SetRange([Clli],[Clli]);
1116        First;
1117        prevDLC:=nil;  ClusterIndex:=0;
1118        if (RecordCount > 0) then
1119          while not eof do
1120            begin
1121              ClusterIndex:=ClusterIndex+1;
1122              DLC             := NewDLC;
1123              DLC^.X          := FieldByName('X').AsInteger;
1124              DLC^.Y          := FieldByName('Y').AsInteger;
1125              DLC^.ClusterID := ClusterIndex;
1126              DLC^.Lat        := FieldByName('Latitude').AsInteger;
1127              DLC^.Long       := FieldByName('Longitude').AsInteger;
1128              if (prevDLC=nil) then pDLC:=DLC
```

```
1129              else prevDLC^.nextDLC:=DLC;
1130              prevDLC:=DLC;
1131              Next;
1132            end;
1133        Close;
1134      end;
1135
1136    // make sure that the previous Demand linked list is disposed before a new
1137    // one is created
1138    if (pData <> nil) then
1139      begin
1140        DisposeData(pData);
1141        pData:=nil;
1142      end;
1143
1144    // create the demand linked list
1145    with dm_ICM.t_Demand do
1146      begin
1147        if not Active then Open;
1148        SetRange([Clli],[Clli]);
1149        First;
1150        prevData:=nil;
1151        if (RecordCount > 0) then
1152          while not eof do
1153            begin
1154              dData            := NewData;
1155              dData^.X         := FieldByName('X').AsInteger;
1156              dData^.Y         := FieldByName('Y').AsInteger;
1157              dData^.ResLines  := FieldByName('Res Lines').AsInteger;
1158              dData^.BusLines  := FieldByName('Bus Lines').AsInteger;
1159              dData^.TotLines  := dData^.ResLines+dData^.BusLines;
1160              dData^.ClusterID := GetClusterId(FieldByName('X Cluster').AsInteger,
1161                                               FieldByName('Y cluster').AsInteger);
1162              AddToClusterLines(dData^.ClusterID,dData^.ResLines,dData^.BusLines,dData^.TotLines);
1163              if (prevData=nil) then pData:=dData
1164              else prevData^.nextData:=dData;
1165              prevData:=dData;
1166              Next;
1167            end;
```

```
1168        Close;
1169      end;
1170
1171    // make sure that the previous Feeder linked list is disposed before a new
1172    // one is created
1173    if (pFeeder <> nil) then
1174      begin
1175        DisposeFeeder(pFeeder);
1176        pFeeder:=nil;
1177      end;
1178
1179    with dm_ICM.t_Arcs do
1180      begin
1181        if not Active then Open;
1182
1183        // create the Feeder linked list
1184        SetRange([Clli,'1'],[Clli,'1']);
1185        First;
1186        prevFeeder:=nil;
1187        if (RecordCount > 0) then
1188          while not eof do
1189            begin
1190              Feeder          := NewFeeder;
1191              Feeder^.xFrom   := FieldByName('From X').AsInteger;
1192              Feeder^.yFrom   := FieldByName('From Y').AsInteger;
1193              Feeder^.xTo     := FieldByName('To X').AsInteger;
1194              Feeder^.yTo     := FieldByName('To Y').AsInteger;
1195              if (prevFeeder=nil) then pFeeder:=Feeder
1196              else prevFeeder^.nextFeeder:=Feeder;
1197              prevFeeder:=Feeder;
1198              Next;
1199            end;
1200
1201        Close;
1202      end;
1203
1204  end; { ReadInAllInfo }
1205
1206  procedure Tf_DemandGUI.tbExitClick(Sender: TObject);
```

```
1207  begin
1208    Close;
1209  end;
1210
1211  procedure Tf_DemandGUI.tbInfoClick(Sender: TObject);
1212  begin
1213    ShowInfoPanel:=not ShowInfoPanel;
1214    pInfo.Visible:=not pInfo.Visible;
1215    if ShowInfoPanel then splitterInfo.Left:=pDemandAndDetail.Width
1216    else splitterInfo.Left:=pDemandAndDetail.Width-5;
1217    tbInfo.Down:=not ShowInfoPanel;
1218  end;
1219
1220  procedure Tf_DemandGUI.tbAspectRatioClick(Sender: TObject);
1221  begin
1222    UseAspectRatio:=not UseAspectRatio;
1223    tbAspectRatio.Down:=UseAspectRatio;
1224    PrevClli:='';
1225    cbCLLIsChange(Sender);
1226  end;
1227
1228  procedure Tf_DemandGUI.tbFeederClick(Sender: TObject);
1229  begin
1230    ShowFeederArcs:=not ShowFeederArcs;
1231    tbFeeder.Down:=ShowFeederArcs;
1232    cbCLLIsChange(Sender);
1233  end;
1234
1235  procedure Tf_DemandGUI.tbDensityMapClick(Sender: TObject);
1236  begin
1237    if (Sender=tbDensityMap) then
1238      begin
1239        tbDensityMap.Enabled:=false;
1240        if not tbClusterMap.Enabled then tbClusterMap.Enabled:=true;
1241        if not tbBlankMap.Enabled then tbBlankMap.Enabled:=true;
1242      end
1243    else if (Sender=tbClusterMap) then
1244      begin
1245        tbClusterMap.Enabled:=false;
```

```
1246        if not tbDensityMap.Enabled then tbDensityMap.Enabled:=true;
1247        if not tbBlankMap.Enabled then tbBlankMap.Enabled:=true;
1248      end
1249    else if (Sender=tbBlankMap) then
1250      begin
1251        tbBlankMap.Enabled:=false;
1252        if not tbDensityMap.Enabled then tbDensityMap.Enabled:=true;
1253        if not tbClusterMap.Enabled then tbClusterMap.Enabled:=true;
1254      end;
1255    cbCLLIsChange(Sender);
1256  end;
1257
1258  procedure Tf_DemandGUI.tbDLCClick(Sender: TObject);
1259  begin
1260    ShowDLC:=not ShowDLC;
1261    tbDLC.Down:=ShowDLC;
1262    cbCLLIsChange(Sender);
1263  end;
1264
1265  procedure Tf_DemandGUI.tbZoomInClick(Sender: TObject);
1266  begin
1267    if (ZoomFactor < MaxZoom) then
1268     begin
1269       pDemand.ScaleBy(110,100);
1270       ZoomFactor := ZoomFactor + 10;
1271       labelZoomFactor.Caption:=IntToStr(ZoomFactor)+'%';
1272     end;
1273  end;
1274
1275  procedure Tf_DemandGUI.tbZoomOutClick(Sender: TObject);
1276  begin
1277    if (ZoomFactor > 10) then
1278     begin
1279       pDemand.ScaleBy(100,110);
1280       ZoomFactor := ZoomFactor - 10;
1281       labelZoomFactor.Caption:=IntToStr(ZoomFactor)+'%';
1282     end;
1283  end;
1284
```

```
1285  procedure Tf_DemandGUI.tbLandscapeClick(Sender: TObject);
1286  begin
1287    isLandscape:=not isLandscape;
1288    tbLandscape.Down:=isLandscape;
1289
1290    // set the proper options
1291    XInches:=11.0;
1292    YInches:=11.0;
1293    if isLandscape then YInches:=8.5
1294    else XInches:=8.5;
1295
1296    // recalc the pixels per inch for each axis
1297    if not UseAspectRatio then
1298      begin
1299        ppDemandPtX:=CalcDemandPtSize(xPPI,yPPI);
1300        ppDemandPtY:=ppDemandPtX;
1301      end
1302    else CalcDemandPtSizeWithAspRatio(xPPI,yPPI,ppDemandPtX,ppDemandPtY);
1303
1304    // Blank out the image
1305    with iDemand.Canvas do
1306      begin
1307        Brush.Color:=clLtGray;
1308        FillRect(Rect(0,0,iDemand.Width,iDemand.Height));
1309      end;
1310
1311    // redraw the information
1312    cbCLLIsChange(Sender);
1313  end;
1314
1315  procedure Tf_DemandGUI.tbPrintClick(Sender: TObject);
1316  var
1317    ppiX,ppiY : integer;
1318    xLeft,xRight,xTop,xBottom,ppDemandPtX,ppDemandPtY,
1319    tickLength,rectWidth,penWidth: integer;
1320    StartPage,EndPage,Page : integer;
1321    OriginalClli : string;
1322    procedure InitMap;
1323    begin
```

```
1324      if not UseAspectRatio then
1325        begin
1326          ppDemandPtX:=CalcDemandPtSize(ppiX,ppiY);
1327          ppDemandPtY:=ppDemandPtX;
1328        end
1329      else CalcDemandPtSizeWithAspRatio(ppiX,ppiY,ppDemandPtX,ppDemandPtY);
1330
1331      rectWidth:=trunc(ppiX*0.275);
1332      tickLength:=trunc(ppiX*0.05);
1333      penWidth:=trunc(ppiX*0.025);
1334
1335      DrawTemplate(ppiX,ppiY,Printer.Canvas,ppDemandPtX,ppDemandPtY,
1336                   xLeft,xRight,xTop,xBottom);
1337
1338      if not tbDensityMap.Enabled then DrawDemandMap(Printer.Canvas,ppDemandPtX,ppDemandPtY,xLeft,xTop)
1339      else if not tbClusterMap.Enabled then
1340  DrawClusterMap(Printer.Canvas,ppDemandPtX,ppDemandPtY,xLeft,xTop);
1341      if not tbDensityMap.Enabled then DrawLegend(ppiX,ppiY,ppDemandPtX,ppDemandPtY,Printer.Canvas);
1342
1343      if ShowFeederArcs then
1344  DrawArcs(ppiX,ppiY,Printer.Canvas,xLeft,xRight,xTop,xBottom,ppDemandPtX,ppDemandPtY,penWidth);
1345      if ShowDLC then PrintDLCs(ppiX,ppiY,Printer.Canvas,xLeft,xRight,xTop,xBottom,ppDemandPtX,ppDemandPtY);
1346    end; { InitMap }
1347  begin
1348    // set up all the options
1349    with PrintDialog1 do
1350      begin
1351        if (cbCLLIs.Items.Count > 0) then
1352          begin
1353            MinPage:=1;  MaxPage:=cbCLLIs.Items.Count;
1354          end
1355        else
1356          begin
1357            MinPage:=0;  MaxPage:=0;
1358          end;
1359        FromPage:=MinPage;  ToPage:=MaxPage;
1360      end;
1361
1362    if PrintDialog1.Execute then
```

```
1363      begin
1364        // get the range specified by the user
1365        if (PrintDialog1.PrintRange=prAllPages) then
1366          begin
1367            StartPage:=PrintDialog1.MinPage;
1368            EndPage:=PrintDialog1.MaxPage;
1369          end
1370        else if (PrintDialog1.PrintRange=prPageNums) then
1371          begin
1372            StartPage:=PrintDialog1.FromPage;
1373            EndPage:=PrintDialog1.ToPage;
1374          end
1375        else
1376          begin
1377            StartPage:=cbCLLIs.ItemIndex+1;
1378            EndPage:=StartPage;
1379          end;
1380
1381        // set the page orientation according to option specified in the vi
1382        if isLandscape then Printer.Orientation:=poLandscape
1383        else Printer.Orientation:=poPortrait;
1384        // set the resolution based on the printer specs
1385        ppiX:=trunc(Printer.PageWidth/xInches);
1386        ppiY:=trunc(Printer.PageHeight/yInches);
1387        // start print job
1388        Printer.BeginDoc;
1389        // cycle through the selection
1390        if (PrintDialog1.PrintRange=prSelection) then InitMap
1391        else
1392          begin
1393            // save the original clli that is on the screen
1394            OriginalClli:=Clli;
1395            for Page:=StartPage to EndPage do
1396              begin
1397                Clli:=cbCLLIs.Items[Page-1];
1398                ReadInAllInfo;
1399                GetMaxMin(true);
1400                InitMap;
1401                if (Page <> EndPage) then Printer.NewPage;
```

```
1402              end;
1403            // restore the original clli info
1404            Clli:=OriginalClli;
1405            ReadInAllInfo;
1406            GetMaxMin(true);
1407          end;
1408        // end print job
1409        Printer.EndDoc;
1410      end;
1411  end;
1412
1413  procedure Tf_DemandGUI.tbHelpClick(Sender: TObject);
1414  begin
1415    GoHelp ('HowDoI.hlp', 351);
1416  end;
1417
1418  procedure Tf_DemandGUI.tbPrevClick(Sender: TObject);
1419  begin
1420    if (Sender=tbPrev) and (cbCLLIs.ItemIndex > 0) then
1421      begin
1422        cbCLLIs.ItemIndex:=cbCLLIs.ItemIndex-1;
1423        cbCLLIsChange(Sender);
1424      end
1425    else if (Sender=tbNext) and (cbCLLIs.ItemIndex < cbCllis.Items.Count) then
1426      begin
1427        cbCLLIs.ItemIndex:=cbCLLIs.ItemIndex+1;
1428        cbCLLIsChange(Sender);
1429      end;
1430  end;
1431
1432  begin
1433   Legends[1].MinValue:=0;
1434   Legends[1].MaxValue:=0;
1435   Legends[1].qColors:=clSilver;
1436   Legends[2].MinValue:=1;
1437   Legends[2].MaxValue:=10;
1438   Legends[2].qColors:=clYellow;
1439   Legends[3].MinValue:=11;
1440   Legends[3].MaxValue:=50;
```

```
1441    Legends[3].qColors:=clAqua;
1442    Legends[4].MinValue:=51;
1443    Legends[4].MaxValue:=100;
1444    Legends[4].qColors:=clLime;
1445    Legends[5].MinValue:=101;
1446    Legends[5].MaxValue:=200;
1447    Legends[5].qColors:=clGreen;
1448    Legends[6].MinValue:=201;
1449    Legends[6].MaxValue:=300;
1450    Legends[6].qColors:=clBlue;
1451    Legends[7].MinValue:=301;
1452    Legends[7].MaxValue:=400;
1453    Legends[7].qColors:=clFuchsia;
1454    Legends[8].MinValue:=400;
1455    Legends[8].MaxValue:=0;
1456    Legends[8].qColors:=clRed;
1457
1458  end.
1459
```

NODES.pas

```
2
3   unit Nodes;
4   {
5
6     This unit contains the routines for the Nodes information. Main function
7     is to maintain table and graph information.
8   }
9
10  interface
11
12  uses
13    SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
14    Forms, Dialogs, StdCtrls, Menus, ExtCtrls, DBCtrls, Grids, DBGrids, DB,
15    DBTables, Assoc, Printers, RepView, TablePrnt, Import, Gen_Exp, Globalda,
16    Query, OSPTypes;
17
18  type
19    Tf_Nodes = class(TForm)
20      t_Nodes: TTable;
21      ds_Nodes: TDataSource;
22      dbg_WireCenter: TDBGrid;
23      dbn_WireCenter: TDBNavigator;
24      MainMenu1: TMainMenu;
25      File1: TMenuItem;
26      New1: TMenuItem;
27      Open1: TMenuItem;
28      N1: TMenuItem;
29      Import1: TMenuItem;
30      N2: TMenuItem;
31      Exit1: TMenuItem;
32      l_Copyright: TLabel;
33      cb_Assoc: TCheckBox;
34      od_WireCenter: TOpenDialog;
35      sd_WireCenter: TSaveDialog;
36      Export1: TMenuItem;
```

```
37      t_CBG: TTable;
38      Print1: TMenuItem;
39      N3: TMenuItem;
40      PrintDialog1: TPrintDialog;
41      Query1: TMenuItem;
42      Form1: TMenuItem;
43      Table1: TMenuItem;
44      procedure FormShow(Sender: TObject);
45      procedure FormResize(Sender: TObject);
46      procedure FormHide(Sender: TObject);
47      procedure Exit1Click(Sender: TObject);
48      procedure New1Click(Sender: TObject);
49      procedure Open1Click(Sender: TObject);
50      procedure Import1Click(Sender: TObject);
51      procedure Export1Click(Sender: TObject);
52      procedure t_NodesAfterPost(DataSet: TDataset);
53      procedure Help1Click(Sender: TObject);
54      procedure Query1Click(Sender: TObject);
55      procedure Print1Click(Sender: TObject);
56      procedure Table1Click(Sender: TObject);
57      procedure Form1Click(Sender: TObject);
58    private
59      { Private declarations }
60
61    public
62      { Public declarations }
63      procedure Entry;
64      procedure PrintNodes( Open : Boolean);
65      procedure PrintFillFactor(Open : Boolean);
66
67    end;
68
69  Const
70    PII = 3.141592653589793;
71    Radius = 20891191.2;
72
73  var
74    f_Nodes: Tf_Nodes;
75    TabName     : String;
```

```
76
77  implementation
78  uses nemain;
79  {$R *.DFM}
80
81  {main entry routine}
82  procedure Tf_Nodes.Entry;
83  begin
84    Show;
85  end;
86
87  {display form event}
88  procedure Tf_Nodes.FormShow(Sender: TObject);
89  begin
90    RunFlag := False;
91
92    cb_Assoc.Checked := False;
93    l_Copyright.Caption := f_associate.GetFileName(5);
94
95    TabName :=  f_associate.GetFileName(2003);
96    With t_Nodes do
97      begin
98         if Active then Close;
99         DatabaseName := ExtractFilePath(TabName);
100        TableName := ExtractFilename(TabName);
101        TableType := ttDefault;
102        Exclusive := True;
103        Open;
104     end;
105   Caption := 'Switching Nodes [' + ExtractFilename(TabName)+']';
106 end;
107
108 {form resize event}
109 procedure Tf_Nodes.FormResize(Sender: TObject);
110 begin
111   if ClientWidth < 350
112     then ClientWidth := 350;
113
114  dbg_WireCenter.Width := ClientWidth-20;
```

```
115   dbn_WireCenter.Width := 191;
116
117   dbg_WireCenter.Left := 10;
118   if ClientWidth > (dbn_WireCenter.Width + cb_Assoc.Width + 5)
119     then dbn_WireCenter.Left := (ClientWidth - (dbn_WireCenter.Width + cb_Assoc.Width + 5)) div 2
120     else dbn_WireCenter.Left := 1;
121   cb_Assoc.Left    := dbn_WireCenter.Left + dbn_WireCenter.Width + 5;
122
123   dbg_WireCenter.Top := l_CopyRight.Height+2;
124   dbn_WireCenter.Top := ClientHeight - (dbn_WireCenter.Height + 5);
125   cb_Assoc.Top :=  dbn_WireCenter.Top + (dbn_WireCenter.Height-cb_Assoc.Height) div 2;
126   dbg_WireCenter.Height :=  dbn_WireCenter.Top - (dbg_WireCenter.Top+5);
127 end;
128
129 {form hide event}
130 procedure Tf_Nodes.FormHide(Sender: TObject);
131 begin
132   if t_Nodes.Active then t_Nodes.close;
133   if t_CBG.Active then t_CBG.close;
134 end;
135
136 {quit}
137 procedure Tf_Nodes.Exit1Click(Sender: TObject);
138 begin
139   if cb_Assoc.Checked
140     then f_associate.SetFileName(2003, TabName);
141  cb_Assoc.Checked := False;
142
143   if RunFlag then f_associate.SetFileName(100, 'YES');
144   Hide;
145 end;
146
147 {create a new empty Node table}
148 procedure Tf_Nodes.New1Click(Sender: TObject);
149 begin
150   sd_WireCenter.Filter := 'Database|*.db';
151
152   if Not sd_WireCenter.execute then exit;
153   TabName := sd_WireCenter.FileName;
```

```
154     cb_Assoc.Checked := True;
155
156     with t_Nodes do
157       begin
158         Active := False;
159         DatabaseName := ExtractFilePath(TabName);
160         TableName := ExtractFilename(TabName);
161         TableType := ttParadox;
162         Exclusive := True;
163         with FieldDefs do
164           begin
165             Clear;
166             Add ('CLLI',              FTString, 11, True);
167             Add ('Host CLLI',         FTString, 11, False);
168             Add ('Switch Type',       FTString, 12, True);
169             Add ('Class',             FTInteger,0,  False);
170             Add ('HML',               FTString, 1,  False);
171             Add ('Latitude',          FTFloat,  0,  False);
172             Add ('Longitude',         FTFloat,  0,  False);
173             Add ('Total Lines',       FTFloat,  0,  False);
174             Add ('Non Switched Lines',FTFloat,  0,  False);  //RFPC0017
175             Add ('Demand Points',     FTInteger,0,  False);
176             Add ('Line Terms',        FTFloat,  0,  False);
177             Add ('BRI',               FTFloat,  0,  False);
178             Add ('PRI',               FTFloat,  0,  False);
179             Add ('PBX',               FTFloat,  0,  False); //RFPC0055
180             Add ('Key',               FTFloat,  0,  False); //RFPC0055
181             Add ('Centranet',         FTFloat,  0,  False); //RFPC0055
182             Add ('Coin',              FTFloat,  0,  False); //RFPC0055
183             Add ('B1',                FTFloat,  0,  False);
184             Add ('Road Feet',         FTFloat,  0,  False);
185             Add ('Road Feet Ratio',   FTFloat,  0,  False);
186             Add ('BedRock',           FTInteger,0,  False);
187             Add ('WaterTable',        FTInteger,0,  False);
188             Add ('Cluster Count 12KFT',FTInteger,0,  False);
189             Add ('Distance Adj',      FTInteger,0,  False);
190             Add ('Width Demand Pt',   FTInteger,0,  False);
191             Add ('Height Demand Pt',  FTInteger,0,  False);
192             Add ('DU',                FTFloat,  0,  False);
```

```
193             Add ('FU',                FTFloat,  0,  False);
194             Add ('F Aerial Pct',      FTFloat,  0,  False);
195             Add ('F Buried Pct',      FTFloat,  0,  False);
196             Add ('F Undgnd Pct',      FTFloat,  0,  False);
197             Add ('C Aerial Pct',      FTFloat,  0,  False);
198             Add ('C Buried Pct',      FTFloat,  0,  False);
199             Add ('C Undgnd Pct',      FTFloat,  0,  False);
200             Add ('Tier B',            FTBoolean,0,  False);
201           end;
202
203         with IndexDefs do
204           begin
205             Clear;
206             Add('Main', 'CLLI', [ixPrimary, ixUnique]);
207             Add('SwType', 'Switch Type', [ixCaseInsensitive]);
208             Add('ByHML', 'HML', [ixCaseInsensitive]);
209           end;
210
211         CreateTable;
212         Open;
213       end;
214     Caption := 'Switching Nodes [' + ExtractFilename(TabName)+']';
215   end;
216
217   {open an existing wirectr table}
218   procedure Tf_Nodes.Open1Click(Sender: TObject);
219   begin
220     od_WireCenter.Filter := 'Database|*.db;*.dbf*';
221
222     if Not od_WireCenter.execute then exit;
223     TabName := od_WireCenter.FileName;
224     With t_Nodes do
225       begin
226          if Active then Close;
227          DatabaseName := ExtractFilePath(TabName);
228          TableName := ExtractFilename(TabName);
229          TableType := ttDefault;
230          Exclusive := True;
231          Open;
```

```
232       end;
233     Caption := 'Switching Nodes [' + ExtractFilename(TabName)+']';
234   end;
235
236   {import a text file}
237   procedure Tf_Nodes.Import1Click(Sender: TObject);
238   begin
239    t_Nodes.Close;
240    ImportDataCall('', AliasPath+'\'+t_Nodes.Tablename, 10);
241    t_Nodes.Open
242   end;
243
244   {export table to text file}
245   procedure Tf_Nodes.Export1Click(Sender: TObject);
246   begin
247     ExportTable(t_Nodes);
248   end;
249
250   {print the table }
251   procedure Tf_Nodes.PrintNodes( Open : Boolean);
252    begin
253      if Open then begin
254        TabName :=  f_associate.GetFileName(2003);
255        With t_Nodes do begin
256          DatabaseName := ExtractFilePath(TabName);
257          TableName := ExtractFilename(TabName);
258          TableType := ttParadox;
259          Exclusive := True;
260          Open;
261        end;
262      end;
263
264      GenericPrintTable(t_Nodes, Nil, 0, False, False, 'Switching Nodes');
265
266      if Open then t_Nodes.Close;
267    end;
268
269   {set the assoc run to true if table was changed}
270   procedure Tf_Nodes.t_NodesAfterPost(DataSet: TDataset);
```

```
271  begin
272    RunFlag := True;
273  end;
274
275  {print the fill factors}
276  procedure Tf_Nodes.PrintFillFactor(Open : Boolean);
277  Const
278      Lab : array[1..2] of String[15] = ('Feeder',
279                                         'Distribution');
280  Var
281    Strg    : String;
282    Output  : Array[1..4] of Real;
283    I       : word;
284    Right   : real;
285    Top     : real;
286    Left    : real;
287    Bottom  : real;
288    LineCnt : word;
289    TFU     : Double;
290    TDU     : Double;
291    LoopF   : Double;
292    LoopD   : Double;
293
294    Procedure Header;
295     Const
296      Label1 : array[1..3] of String[31] = ('CLLI Code',
297                                            'Feeder Fill',
298                                            'Distribution Fill'
299                                            );
300     Var
301       I    : word;
302       rTop : Real;
303       rBottom : Real;
304       rLeft : Real;
305       rRight : Real;
306
307     begin
308     Inc(Page);
309  { Print Heading }
```

```
310       PageObj.FntNam  := 'Arial';
311       PageObj.FntSize := 14;
312       PageObj.FntStyle:= [fsBold];
313       PageObj.Color := clBlue;
314
315       f_ReportView.PrintText(Page, 0.5,0.25, 6.0,0.45, 'FILL FACTORS', dt_Center+dt_SingleLine);
316
317       PageObj.FntSize := 10;
318       PageObj.FntStyle:= [];
319
320       f_ReportView.PrintText(Page, 6.0,0.25, 8.0,0.45, 'State of '+f_associate.GetFileName(200),
321   dt_Left+dt_SingleLine);
322       case Asmp.CU_Loop_Length of
323   //RFPC0068
324        12 : f_ReportView.PrintText(Page, 6.0,0.45, 8.0,0.65, 'Option: 12 KFT', dt_Left+dt_SingleLine);
325   //RFPC0068
326        15 : f_ReportView.PrintText(Page, 6.0,0.45, 8.0,0.65, 'Option: 12 KFT with ADSL',
327   dt_Left+dt_SingleLine);  //RFPC0068
328        18 : f_ReportView.PrintText(Page, 6.0,0.45, 8.0,0.65, 'Option: 18 KFT', dt_Left+dt_SingleLine);
329   //RFPC0068
330       end;
331   //RFPC0068
332
333       PageObj.Color := clBlack;
334       f_ReportView.PrintText(Page, 0.5, 0.85, 8.0,1.1, f_associate.GetFileName(201), dt_Left+dt_SingleLine);
335
336       PageObj.Color := clBlue;
337       f_ReportView.PrintBox(Page, 0.5,1.1, 7.0,1.4, 1, clBlue, clWhite); //RFPC0006
338       rTop := 1.1;
339       rBottom := 1.4;
340       rLeft := 0.5;
341       rRight := 3.0;
342       f_ReportView.PrintText(Page, rLeft,rTop+0.025, rRight,rBottom, Label1[1], dt_Center+dt_WordBreak);
343       for I := 1 to 2 do
344         begin
345           f_ReportView.PrintLine(Page, rRight,rTop,rRight,rBottom, 1);
346           rLeft := rRight;
347           rRight := rLeft + 2.0;
```

```
348          f_ReportView.PrintText(Page, rLeft,rTop+0.025, rRight,rBottom, Label1[I+1],
349  dt_Center+dt_WordBreak);
350        end;
351    end;
352
353   procedure Footer;
354    begin
355      PageObj.FntSize  := 10;
356      PageObj.Color := clBlue;
357      PageObj.FntStyle := [fsBold, fsItalic];
358
359      Strg := f_associate.GetFileName(6);
360      f_ReportView.PrintText(Page, 0.5,10.5, 8.0,10.7, Strg, dt_Center+dt_SingleLine);
361
362      PageObj.FntSize  := 10;
363      PageObj.Color := clBlack;
364      PageObj.FntStyle := [];
365      Strg := FormatDateTime('mmmm dd, yyyy  hh:mm am/pm', Now);
366      f_ReportView.PrintText(Page, 0.5,10.5, 2.75,10.7, Strg, dt_Left+dt_SingleLine);
367
368      Strg := f_associate.GetFileName(202);
369      if Strg <> ''
370        then begin
371               f_ReportView.PrintText(Page, 0.5,10.3, 8.0,10.5, Strg, dt_Left+dt_SingleLine);
372               f_ReportView.PrintLine(Page, 0.5,10.2, 8.0, 10.2, 2);
373             end
374        else f_ReportView.PrintLine(Page, 0.5,10.4, 8.0, 10.4, 2);
375  end;
376
377   begin
378     if Open
379        then begin
380               TabName :=  f_associate.GetFileName(2003);
381               With t_Nodes do
382                 begin
383                   DatabaseName := ExtractFilePath(TabName);
384                   TableName := ExtractFilename(TabName);
385                   TableType := ttParadox;
386                   Exclusive := True;
```

```
387                 Open;
388                 IndexName := '';
389               end;
390             end;
391
392   Header;
393   f_ReportView.AddBookMark(0, Page, 'OSP Fill Factors' ); //RFPC0006
394   LineCnt := 0;
395   Bottom  := 1.45;
396   TFU     := 0.0;
397   TDU     := 0.0;
398   LoopF   := 0.0;
399   LoopD   := 0.0;
400
401   while not t_Nodes.EOF do
402      begin
403        if LineCnt > 51
404          then begin
405                 Footer;
406                 Header;
407                 LineCnt := 0;
408                 Bottom := 1.45;
409               end;
410
411        Inc (LineCnt);
412
413        OutPut[1] := t_Nodes.FieldByName('FU').AsFloat;
414        OutPut[2] := t_Nodes.FieldByName('DU').AsFloat;
415        TFU       := TFU + OutPut[1] * t_Nodes.FieldByName('Total Lines').AsFloat;
416        TDU       := TDU + OutPut[2] * t_Nodes.FieldByName('Total Lines').AsFloat;
417        if OutPut[1] > 0
418          then LoopF := LoopF + t_Nodes.FieldByName('Total Lines').AsFloat;
419        if OutPut[2] > 0
420          then LoopD := LoopD + t_Nodes.FieldByName('Total Lines').AsFloat;
421
422        PageObj.FntSize := 10;
423        PageObj.Color := clBlack;
424
425        Left   := 0.5 ;
```

```
426          Right  := 3.0;
427          Top    := Bottom;
428          Bottom := Top + 0.16667;
429          f_ReportView.PrintText(Page, Left,Top, Right,Bottom, t_Nodes.FieldByName('CLLI').AsString,
430                                 dt_Left+dt_SingleLine);
431          for i := 1 to 2 do
432             begin
433               Left   := Right;
434               Right  := Left + 2.0;
435               Strg   := Format('%5.2f', [ Output[I]*100.0]);
436               f_ReportView.PrintText(Page, Left,Top, Right,Bottom, Strg, dt_Right+dt_SingleLine);
437             end;
438         t_Nodes.Next;
439       end;
440
441     OutPut[1] := TFU/LoopF;
442     OutPut[2] := TDU/LoopD;
443
444     PageObj.FntSize := 10;
445     PageObj.Color := clBlack;
446
447     Left   := 0.5 ;
448     Right  := 3.0;
449     Top    := Bottom + 0.1;
450     Bottom := Top + 0.16667;
451     f_ReportView.PrintText(Page, Left,Top, Right,Bottom, 'State Average',
452                            dt_Left+dt_SingleLine);
453     for i := 1 to 2 do
454        begin
455          Left   := Right;
456          Right  := Left + 2.0;
457          Strg   := Format('%5.2f', [ Output[I]*100.0]);
458          f_ReportView.PrintText(Page, Left,Top, Right,Bottom, Strg, dt_Right+dt_SingleLine);
459        end;
460
461     Footer;
462     if Open then t_Nodes.Close;
463    end;
464
```

```
465  procedure Tf_Nodes.Help1Click(Sender: TObject);
466  begin
467  end;
468
469  {call query module}
470  procedure Tf_Nodes.Query1Click(Sender: TObject);
471  begin
472    f_Query.Query(t_Nodes);
473  end;
474
475  procedure Tf_Nodes.Print1Click(Sender: TObject);
476  begin
477    GenericPrintTable(t_Nodes, Nil, 0, False, True, 'Switching Nodes');
478  end;
479
480  {call help module}
481  procedure Tf_Nodes.Table1Click(Sender: TObject);
482  begin
483     GoHelp ('Datatables.hlp', 412);
484  end;
485
486  procedure Tf_Nodes.Form1Click(Sender: TObject);
487  begin
488    GoHelp ('MenuIt.hlp', 327);
489  end;
490
491  end.  {end unit}
492
```

OUTSIDE PLANT (OSP).pas

```
2
3   unit osp;
4
5   interface
6
7   uses
8     Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
9     StdCtrls, ExtCtrls, ComCtrls, Tabnotbk, Buttons, Db, DBTables,
10    OSPTypes, Distrib, Spanning, Math;
11
12    procedure LoadParameters;
13
14
15  implementation
16
17  uses SKCluster, ICMDM, Demand, Place, OSPInvest, IOFArc, Interofc, Invntry;
18
19
20  procedure LoadParameters;
21   Var
22     obscount : Integer;
23     I,N,K    : Integer;
24     TLines   : Integer;
25     clli     : String;
26
27   begin
28     DM_ICM.t_Nodes.First;
29      while Not DM_ICM.t_Nodes.Eof do  // Loop through Node table one record a time
30        begin
31          f_Demand.Gauge1.Progress := Ceil((DM_ICM.t_Nodes.RecNo-1) / DM_ICM.t_Nodes.RecordCount * 100.0);
32          f_Demand.l_Progress.Caption := Format('Clustering %d of %d', [DM_ICM.t_Nodes.RecNo-1,
33  DM_ICM.t_Nodes.RecordCount]);
34          Application.ProcessMessages;
35
36          Asmp.HML := DM_ICM.t_Nodes.FieldByName('HML').AsString;  // High, medium and Low density
```

```
37          if Asmp.HML = 'L'
38           then  Asmp.Structure_Flag := 0
39           else  Asmp.Structure_Flag := 1;
40
41          CLLI := DM_ICM.t_Nodes.FieldByName('CLLI').AsString; // Set CLLI code
42          DM_ICM.t_Demand.SetRange([CLLI], [CLLI]);           // Set Filter range on demand table
43          DM_ICM.t_Demand.First;                              // Start at first record of demand table
44
45          // initialize variables
46          FeedCable:= 0;
47          DemandList.Clear;
48          DemandList.Capacity := MaxObs;
49
50          //Wirecenter level Feeder Variables
51          Pb.clli          := DM_ICM.t_Nodes.FieldByName('clli').AsString;
52          Pb.LocDist[0]    := DM_ICM.t_Nodes.FieldByName('Width Demand Pt').AsFloat;
53          Pb.LocDist[1]    := DM_ICM.t_Nodes.FieldByName('Height Demand Pt').AsFloat;
54          Pb.AdjMaxLoop    := Asmp.MaxLoop - DM_ICM.t_Nodes.FieldByName('Distance Adj').AsInteger;
55          RoadFeet         := DM_ICM.t_Nodes.FieldByName('Road Feet').AsInteger;
56          F_Percent_Aerial := DM_ICM.t_Nodes.FieldByName('F Aerial Pct').AsFloat;
57          F_Percent_Buried := DM_ICM.t_Nodes.FieldByName('F Buried Pct').AsFloat;
58          F_Percent_Undgnd := DM_ICM.t_Nodes.FieldByName('F Undgnd Pct').AsFloat;
59          C_Percent_Aerial := DM_ICM.t_Nodes.FieldByName('C Aerial Pct').AsFloat;
60          C_Percent_Buried := DM_ICM.t_Nodes.FieldByName('C Buried Pct').AsFloat;
61          C_Percent_Undgnd := DM_ICM.t_Nodes.FieldByName('C Undgnd Pct').AsFloat;
62          if DM_ICM.t_Nodes.FieldByName('Tier B').AsBoolean
63            then Tier_Flag := 'B'
64            else Tier_Flag := 'A';
65
66          Res_Total := 0;
67          Bus_Total := 0;
68          BedRock   := 0;
69          WaterTable:= 0;
70          Elevation := 0;
71          Highway   := 0;
72          obscount  := 0;
73
74          if DM_ICM.t_Demand.RecordCount > 0
75           then while Not DM_ICM.t_Demand.Eof do  // Loop through demand table for Clustering
```

```
76            begin
77              if (DM_ICM.t_Demand.FieldByName('Res Lines').AsInteger +
78                  DM_ICM.t_Demand.FieldByName('Bus Lines').AsInteger) > 0
79                then begin   // select only records which contain demand
80                       //This begins the list of items that are updated with each record
81                       Inc(obscount);
82                       //Updates Feed Detail Records
83                       New(DemandPt);
84                       DemandPt^.X          := DM_ICM.t_Demand.FieldByName('x').AsInteger;
85                       DemandPt^.Y          := DM_ICM.t_Demand.FieldByName('y').AsInteger;
86                       DemandPt^.XinFt      := DemandPt^.X * round(Pb.LocDist[0]);
87                       DemandPt^.YinFt      := DemandPt^.Y * round(Pb.LocDist[1]);
88                       DemandPt^.Res_Lines := DM_ICM.t_Demand.FieldByName('Res Lines').AsInteger;
89                       DemandPt^.Bus_Lines := DM_ICM.t_Demand.FieldByName('Bus Lines').AsInteger;
90                       DemandList.Add(DemandPt);
91
92                       Res_Total  := Res_Total  + DM_ICM.t_Demand.FieldByName('Res Lines').AsInteger;
93                       Bus_Total  := Bus_Total  + DM_ICM.t_Demand.FieldByName('Bus Lines').AsInteger;
94                       BedRock    := BedRock    + DM_ICM.t_Demand.FieldByName('BedRock').AsInteger;
95                       WaterTable := WaterTable + DM_ICM.t_Demand.FieldByName('Water Table').AsInteger;
96                       Elevation  := Elevation  + DM_ICM.t_Demand.FieldByName('Elevation Code').AsInteger;
97                       if DM_ICM.t_Demand.FieldByName('Highway Flag').AsBoolean
98                         then Inc(Highway);
99                     end;
100             DM_ICM.t_Demand.Next;
101           end;
102
103         Pb.NumObs := obscount;
104
105         if obscount = 0
106           then begin
107                  DM_ICM.t_DLC.Append;
108                  DM_ICM.t_DLC.FieldByName('CLLI').AsString     := CLLI;
109                  DM_ICM.t_DLC.FieldByName('X').AsInteger       := 0;
110                  DM_ICM.t_DLC.FieldByName('Y').AsInteger       := 0;
111                  DM_ICM.t_DLC.FieldByName('Longitude').AsFloat :=
112                           DM_ICM.t_Nodes.FieldByName('Longitude').AsFloat;
113                  DM_ICM.t_DLC.FieldByName('Latitude').AsFloat  :=
114                           DM_ICM.t_Nodes.FieldByName('Latitude').AsFloat;
```

```
115                 DM_ICM.t_DLC.Post;
116               end;
117
118         if obscount > 0
119          then begin
120                 // Normalize flags
121               BedRock    := round(BedRock/Pb.NumObs);
122               WaterTable:= round(WaterTable/Pb.NumObs);
123               Elevation := round(Elevation/Pb.NumObs);
124               Highway    := round(Highway/Pb.NumObs);
125
126               // Update total lines, BedRock and WaterTable fields in the nodes table
127               DM_ICM.t_Nodes.Edit;
128               DM_ICM.t_Nodes.FieldByName('Total Lines').AsInteger := Res_Total + Bus_Total;
129               DM_ICM.t_Nodes.FieldByName('BedRock').AsInteger     := BedRock;
130               DM_ICM.t_Nodes.FieldByName('WaterTable').AsInteger  := WaterTable;
131               DM_ICM.t_Nodes.Post;
132
133               // Cluster wire center
134               LastClusterCountAdj := 0;
135               repeat
136                 K_Cluster (DemandList, obscount, Pb.NumClusters, Pb.AdjMaxLoop);
137                 Application.ProcessMessages;
138               until GoodResult(DemandList, obscount, Pb.NumClusters, Pb.AdjMaxLoop);
139
140               Sai.clli       := clli;
141               Sai.gridweight := Pb.LocDist[0]/Pb.LocDist[1];
142               Sai.LatDist    := Pb.LocDist[1];
143
144               Res_Total := 0;
145               Bus_Total := 0;
146               Sai.maxSAIs    := Pb.NumClusters;
147               Sai.Rtype      := 'F';
148
149               if Pb.NumClusters > 1  // check to see if there is more than the core cluster
150                 then begin           // if so, setup data and span clusters
151                        SAIList.Clear;
152                        SAIList.Capacity := MaxObs;
153
```

```
154                          I := 0;
155                          K := 0;
156                          while I < Pb.NumClusters do
157                            begin
158                              if (K_Count[I] > 0) or (I=0)
159                                then begin
160                                       // Add office  and DLC locations to DLC table
161                                       DM_ICM.t_DLC.Append;
162                                       DM_ICM.t_DLC.FieldByName('CLLI').AsString     := CLLI;
163                                       DM_ICM.t_DLC.FieldByName('X').AsInteger       := Round(K_Means[I].X);
164                                       DM_ICM.t_DLC.FieldByName('Y').AsInteger       := Round(K_Means[I].Y);
165                                       DM_ICM.t_DLC.FieldByName('Longitude').AsFloat :=
166  Round(K_Means[I].X)/200.0 +
167                                              DM_ICM.t_Nodes.FieldByName('Longitude').AsFloat;
168                                       DM_ICM.t_DLC.FieldByName('Latitude').AsFloat  :=
169  Round(K_Means[I].Y)/200.0 +
170                                              DM_ICM.t_Nodes.FieldByName('Latitude').AsFloat;
171                                       DM_ICM.t_DLC.Post;
172
173                                       New(SaiNode);
174                                       SaiNode^.Node := K;
175                                       SaiNode^.X    := Round(K_Means[I].X);
176                                       SaiNode^.Y    := Round(K_Means[I].Y);
177                                       SaiNode^.XinFt:= Round(K_Means[I].X) * round(Pb.LocDist[0]);
178                                       SaiNode^.YinFt:= Round(K_Means[I].Y) * round(Pb.LocDist[1]);
179                                       SaiNode^.SiteLines := 0;
180                                       SaiNode^.Link := 0;
181                                       SaiNode^.Axis := 0;
182
183                                       For N := 0 to obscount-1 do
184                                         begin
185                                           DemandPt := DemandList.Items[N];
186                                           if DemandPt^.Cluster = I
187                                             then begin
188                                                    SaiNode^.SiteLines := SaiNode^.SiteLines +
189  DemandPt^.Bus_Lines +
190                                                                          DemandPt^.Res_Lines;
191                                                    if I > 0
```

```
                                                then begin   // sum up Bus and Res line outside to core
cluster
                                                      Res_Total := Res_Total  + DemandPt^.Res_Lines;
                                                      Bus_Total := Bus_Total  + DemandPt^.Bus_Lines;
                                                    end;
                                              end;
                                        end;
                                      SAIList.Add(SaiNode);
                                      Inc(K);
                                    end;
                              Inc(I);
                            end;

                          Sai.maxSAIs := K;
                          if Asmp.MaxLoop = 12000
                            then begin
                                   // Update Cluster count in the nodes table
                                   DM_ICM.t_Nodes.Edit;
                                   DM_ICM.t_Nodes.FieldByName('Cluster Count 12KFT').AsInteger :=
Sai.maxSAIs;
                                   DM_ICM.t_Nodes.Post;
                                 end;

                          // Update demand table
                          I := 0;
                          while I < Obscount do
                            begin
                              DemandPt := DemandList.Items[I];
                              if DM_ICM.t_Demand.FindKey([CLLI, DemandPt^.x, DemandPt^.y])
                                then begin
                                       // Update demand table with cluster information
                                       DM_ICM.t_Demand.Edit;
                                       DM_ICM.t_Demand.FieldByName('X Cluster').AsInteger :=
                                              Round(K_Means[DemandPt^.Cluster].X);
                                       DM_ICM.t_Demand.FieldByName('Y Cluster').AsInteger :=
                                              Round(K_Means[DemandPt^.Cluster].Y);
                                       DM_ICM.t_Demand.Post;
                                     end;
                              Inc(I);
```

```
231                         end;
232
233
234                       Span;  // span clusters
235                     end
236                  else begin
237                       if Asmp.MaxLoop = 12000
238                        then begin
239                               // Update Cluster count in the nodes table
240                               DM_ICM.t_Nodes.Edit;
241                               DM_ICM.t_Nodes.FieldByName('Cluster Count 12KFT').AsInteger := Sai.maxSAIs;
242                               DM_ICM.t_Nodes.Post;
243                             end;
244                       // Add office location to DLC table
245                       DM_ICM.t_DLC.Append;
246                       DM_ICM.t_DLC.FieldByName('CLLI').AsString     := CLLI;
247                       DM_ICM.t_DLC.FieldByName('X').AsInteger       := 0;
248                       DM_ICM.t_DLC.FieldByName('Y').AsInteger       := 0;
249                       DM_ICM.t_DLC.FieldByName('Longitude').AsFloat :=
250                                DM_ICM.t_Nodes.FieldByName('Longitude').AsFloat;
251                       DM_ICM.t_DLC.FieldByName('Latitude').AsFloat  :=
252                                DM_ICM.t_Nodes.FieldByName('Latitude').AsFloat;
253                       DM_ICM.t_DLC.Post;
254                     end;
255
256               For I := 0 to Pb.NumClusters-1 do
257                if (K_Count[I] > 0) or (I = 0)
258                  then begin
259                         Res_Total := 0;
260                         Bus_Total := 0;
261                         BedRock   := 0;
262                         WaterTable:= 0;
263                         Elevation := 0;
264                         Highway   := 0;
265                         Sai.maxSAIs := 0;
266                         if I = 0
267                           then Sai.Rtype := 'W'
268                           else Sai.Rtype := 'S';
269
```

```
270                         SAIList.Clear;
271                         SAIList.Capacity := MaxObs;
272
273                         New(SaiNode);
274                         SaiNode^.Node := 0;
275                         SaiNode^.X    := Round(K_Means[I].X);
276                         SaiNode^.Y    := Round(K_Means[I].Y);
277                         SaiNode^.XinFt:= Round(K_Means[I].X) * round(Pb.LocDist[0]);
278                         SaiNode^.YinFt:= Round(K_Means[I].Y) * round(Pb.LocDist[1]);
279                         SaiNode^.SiteLines := 0;
280                         SaiNode^.Link := 0;
281                         SaiNode^.Axis := 0;
282                         SAIList.Add(SaiNode);
283
284                         TLines := 0;
285                         For N := 0 to obscount-1 do
286                           begin
287                             DemandPt := DemandList.Items[N];
288                             if DemandPt^.Cluster = I
289                               then begin
290                                      TLines     := TLines + DemandPt^.Bus_Lines + DemandPt^.Res_Lines;
291                                      Bus_Total  := Bus_Total  + DemandPt^.Bus_Lines;
292                                      Res_Total  := Res_Total  + DemandPt^.Res_Lines;
293                                      if DM_ICM.t_Demand.FindKey([SAI.Clli, DemandPt^.X, DemandPt^.Y])
294                                       then begin
295                                              BedRock    := BedRock    +
296
297   DM_ICM.t_Demand.FieldByName('BedRock').AsInteger;
298                                              WaterTable := WaterTable +
299                                                            DM_ICM.t_Demand.FieldByName('Water
300   Table').AsInteger;
301                                              Elevation  := Elevation  +
302                                                            DM_ICM.t_Demand.FieldByName('Elevation
303   Code').AsInteger;
304                                              if DM_ICM.t_Demand.FieldByName('Highway Flag').AsBoolean
305                                                then Inc(Highway);
306                                            end;
307
```

```
308                                     If Not ((DemandPt^.X = Round(K_Means[I].X)) and (DemandPt^.Y =
309 Round(K_Means[I].Y)))
310                                       then begin
311                                             Inc(Sai.maxSAIs);
312
313                                             New(SaiNode);
314                                             SaiNode^.Node := 0;
315                                             SaiNode^.X    := DemandPt^.X;
316                                             SaiNode^.Y    := DemandPt^.Y;
317                                             SaiNode^.XinFt:= DemandPt^.XinFt;
318                                             SaiNode^.YinFt:= DemandPt^.YinFt;;
319                                             SaiNode^.Link := 0;
320                                             SaiNode^.Axis := 0;
321                                             SaiNode^.SiteLines := DemandPt^.Bus_Lines +
322 DemandPt^.Res_Lines;
323                                             SAIList.Add(SaiNode);
324                                           end ;
325                                   end;
326                             end;
327
328                           SaiNode := SAIList.Items[0];
329                           SaiNode^.SiteLines := TLines;
330                           Inc(Sai.maxSAIs);
331
332                           // Normalize flags
333                           if K_Count[I] > 0
334                             then begin
335                                    BedRock   := round(BedRock/K_Count[I]);
336                                    WaterTable:= round(WaterTable/K_Count[I]);
337                                    Elevation := round(Elevation/K_Count[I]);
338                                    Highway   := round(Highway/K_Count[I]);
339                                  end;
340                           Span;
341                           if Sai.Rtype = 'W'
342                             then begin
343                                    OSP_Table.Write (Sai.CLLI, '000000', 'TDCLOOP', 'D', Res_Total+Bus_Total,
344                                                     Res_Total+Bus_Total,  Res_Total+Bus_Total);
345                                    OSP_Table.Write (Sai.CLLI, '000000', 'RDCLOOP', 'D', Res_Total,
346                                                     Res_Total, Res_Total);
```

```
347                                 OSP_Table.Write (Sai.CLLI, '000000', 'BDCLOOP', 'D', Bus_Total,
348                                                  Bus_Total, Bus_Total);
349                                 // Update lines terms (core lines) in the nodes table
350                                 DM_ICM.t_Nodes.Edit;
351                                 DM_ICM.t_Nodes.FieldByName('Line Terms').AsInteger :=
352 Res_Total+Bus_Total;
353                                 DM_ICM.t_Nodes.Post;
354                               end
355                            else begin
356                                 OSP_Table.Write (Sai.CLLI, '000000', 'TDFLOOP', 'D', Res_Total+Bus_Total,
357                                                  Res_Total+Bus_Total,  Res_Total+Bus_Total);
358                                 OSP_Table.Write (Sai.CLLI, '000000', 'RDFLOOP', 'D', Res_Total,
359                                                  Res_Total, Res_Total);
360                                 OSP_Table.Write (Sai.CLLI, '000000', 'BDFLOOP', 'D', Bus_Total,
361                                                  Bus_Total, Bus_Total);
362                                 SaiNode := SAIList.Items[0];
363                                 PairGainInvest(Res_Total+Bus_Total, Res_Total, Bus_Total);
364                               end;
365                     end;
366
367                  RoadFtAdj   := 1.0 - FeedCable/RoadFeet; // calc percent remaining RF not used
368
369                  DM_ICM.t_Demand.First;
370                  while Not DM_ICM.t_Demand.Eof do  // Loop through demand table for Distribution
371                    begin
372                      if (DM_ICM.t_Demand.FieldByName('Res Lines').AsInteger +
373                          DM_ICM.t_Demand.FieldByName('Bus Lines').AsInteger) > 0
374                        then begin   // select only records which contain demand
375                               // load data for distribution
376                               DistValues.CLLI            := Sai.CLLI;
377                               DistValues.Cluster.X       := DM_ICM.t_Demand.FieldByName('X
378 Cluster').AsInteger;  // Cluster X
379                               DistValues.Cluster.Y       := DM_ICM.t_Demand.FieldByName('Y
380 Cluster').AsInteger;  // Cluster Y
381                               DistValues.Demand.X        := DM_ICM.t_Demand.FieldByName('X').AsInteger;
382                               DistValues.Demand.Y        := DM_ICM.t_Demand.FieldByName('Y').AsInteger;
383                               DistValues.Bus_Lines       := DM_ICM.t_Demand.FieldByName('Bus
384 Lines').AsInteger;
```

```
385                              DistValues.Res_Lines       := DM_ICM.t_Demand.FieldByName('Res
386   Lines').AsInteger;
387                              DistValues.road_feet       := DM_ICM.t_Demand.FieldByName('Road
388   Length').AsInteger;
389                              BedRock                    := DM_ICM.t_Demand.FieldByName('BedRock').AsInteger;
390                              WaterTable                 := DM_ICM.t_Demand.FieldByName('Water
391   Table').AsInteger;
392                              Elevation                  := DM_ICM.t_Demand.FieldByName('Elevation
393   Code').AsInteger;
394                              if DM_ICM.t_Demand.FieldByName('Highway Flag').AsBoolean
395                                then Highway := 1
396                                else Highway := 0;
397
398                              Distribution; // Call Distribution
399                            end;
400                       DM_ICM.t_Demand.Next;
401                    end;
402                end;
403
404           DM_ICM.t_Nodes.Next;  // move to next wire center
405         end;
406
407       ProcessIArcFile;     //Update the interoffice related arc info
408       f_InterOfc.Process;  //InterOffice investments
409       Placement;           //Determine Arc Placement
410       ArcInvestments;      //Cable and placement investment;
411       f_InterOfc.ProcessRingDistance;  //InterOffice ring distances
412    end;
413  end.
```

OSPINVEST.pas

```
2
3   unit OSPInvest;
4
5   interface
6
7     function  Trench(Clli : String; ArcType, Xpos, Ypos : integer;
8                           Length : Double) : Double;
9     function  Conduit(Clli : String; ArcType, Xpos, Ypos, Demand, Bus_Lines : integer;
10                          Length : Double; InvType: String)  : Double;
11    function  Poles (Clli : String; ArcType, Xpos, Ypos : integer;
12                          Length : Double) : Double;
13
14    function  PairGainInvest(TDemand, RDemand, BDemand : Integer): Integer;
15    function  XConnInvest(Clli : String; X, Y, TD : Integer) : Double;
16
17    function  Place_CU_Aerial(Clli : String; ArcType, Xpos, Ypos : integer;
18                          Length, Demand  : Longint; UFill : Double) : Double;
19    function  Place_CU_Buried(Clli : String; ArcType, Xpos, Ypos : integer;
20                          Length, Demand  : Longint; UFill : Double) : Double;
21    function  Place_CU_UndGnd(Clli : String; ArcType, Xpos, Ypos : integer;
22                          Length, Demand  : Longint; UFill : Double) : Double;
23    function  Place_FI_Aerial(Clli : String; ArcType, Xpos, Ypos : integer;
24                          Length, Strands, Demand : Longint; UFill : Double) : Double;
25    function  Place_FI_Buried(Clli : String; ArcType, Xpos, Ypos : integer;
26                          Length, Strands, Demand : Longint; UFill : Double) : Double;
27    function  Place_FI_UndGnd(Clli : String; ArcType, Xpos, Ypos : integer;
28                          Length, Strands, Demand : Longint; UFill : Double) : Double;
29
30    procedure ArcInvestments;
31
32   Var
33
34
35    PercntHand       : Double;       // Percent Hand Dig for high density urban areas is defined in the
36  template.
```

```
37    PercntConcrete    : Double;       // Percent Concrete for high density urban areas is defined in the
38  template.
39    PerExtend         : Integer;      //number of loop extenders
40
41  implementation
42    Uses
43      OSPLabor, OSPMater, OSPTypes, Invntry, Dialogs, SysUtils, ICMDM, Demand,
44      Interofc, Monitr, forms, Assoc, dbtables, Math;
45
46   Var
47    OSPType           : String[1];
48
49    Cable_Cnt         : Integer;
50    Splice_Pts        : Integer;
51    Num_Of_Ducts      : Integer;
52
53    Sharing_Buried    : Double;
54    Sharing_Aerial    : Double;
55    Sharing_Undgnd    : Double;
56    Foreign_Poles     : Double;
57
58    function Trench(Clli : String; ArcType, Xpos, Ypos : integer;
59                            Length : Double) : Double;
60
61    { The Trenching routine does the trenching for buried feeder distribution and
62      interoffice. It does not do the trenching for conduit systems.  It does both
63      plowing and trenching.
64
65          Labor Activity Codes:  (used in Trenching investment calculation)
66          LP43C:      Place Fiber Cable in Conduit
67          LP51A:      Plow Cable - Short Distance (Up to 1000' - 30" Depth)
68          LP51B:      Plow Cable - Long Distance (Greater than 1000' - 30" Depth)
69          LP51C:      Plow Cable - Additional Depth (Each additional 6" increment)
70          LP52A:      Pre-ripping
71          LP54A:      Trench - Short Distance (up to 1000' - 30" depth)
72          LP54B:      Trench - Long Distance (greater than 1000' - 30" depth)
73          LP54C:      Trenching - Additional Depth (each additional 6" depth increment)
74          LP57A:      Hand Dig Trench
75          LP59A:      Bore/Pull Cable (up to 4")
```

```
76          LP70B:      Rock Saw
77          LP70C:      Plow Additional Cable/Split Duct
78          LP93C:      Cut and Remove Concrete (0 to 100 square feet)
79          LP93D:      Place Concrete (0 to 100 square feet)
80      }
81
82      Var
83        Trench_inv : Double;          //  value that is returned - total cost of trenching
84        Per_HD     : Double;          //  percent hand dig
85        Per_Con    : Double;          //  percent concrete
86        Per_Bor    : Double;          //  percent boring
87        Size       : Integer;         //  the depth of plowing/trenching required (inches)
88
89      begin
90        Case ArcType of
91           0 : begin
92                 OSPType := 'I';
93                 Sharing_Buried := Asmp.Sharing_F_Buried;
94               end;
95           1 : begin
96                 OSPType := 'F';
97                 Sharing_Buried := Asmp.Sharing_F_Buried;
98               end;
99        2,3 : begin
100                if DM_ICM.t_Arcs.FieldByName('OSP Type').AsString = 'D'
101                  then begin
102                         OSPType := 'D';
103                         Sharing_Buried := Asmp.Sharing_D_Buried;
104                       end
105                  else begin
106                         OSPType := 'F';
107                         Sharing_Buried := Asmp.Sharing_F_Buried;
108                       end;
109              end;
110          4 : begin
111                OSPType := 'D';
112                Sharing_Buried := Asmp.Sharing_D_Buried;
113              end
114        end;
```

```
115
116       {test to see if there is buried cable length for this placement}
117       if Length < 1 then begin
118         Result := 0;
119         exit;    // if not, then exit
120       end;
121
122     { Structure_Flag is assigned one of three values: 0, 1, 2.  The assignment can
123       either be done from the Feeder routine or from the templates. A value of 0 in
124       the feeder routine corresponds to a density of less than 50 lines per square mile.
125       A value of 1 corresponds to all other densities.  A value of 2 is assigned in the
126       templates. This corresponds to the user defined high density of business lines.}
127       // Case statement change per Roger White
128      Case Asmp.Structure_Flag  of
129        0: begin                 // desity < 50 lines per square mile
130               Per_HD  := 0.0;
131               Per_Con := 0.0;
132               Per_Bor := 0.0;
133            end;
134        1: begin                 // user input values where density > 50 lines per sq mile
135               Per_HD  := Asmp.Percent_Hand;        // Percent Hand Dig is defined under General User Options
136   for Outside Plant
137               Per_Con := Asmp.Percent_Concrete;    // Percent Concrete is defined under General User Options
138   for Outside Plant
139               Per_Bor := Asmp.Percent_Bore;        // Percent Bore is defined under General User Options for
140   Outside Plant
141           end;
142        2: begin                   // applied when # business units > limit set in grid
143               Per_HD  := Asmp.Percent_Hand;        // Percent Hand Dig is defined under General User Options
144   for Outside Plant
145               Per_Con := 0.80;    // Percent Concrete for high density urban areas is defined in the template.
146               Per_Bor := 0.0;
147           end;
148      end;
149
150    { This section calculates the plowing and trenching investments for buried
151      distribution cables.  Random separation is assumed in all distribution plowing
152      and trenching - the cables do not have to be separated by a fix distance such as
153      12" because the power cables are assumed to be carrying lower voltage.
```

```
154
155     ProcessID is used to track what section of program is calling the trenching
156     routine.  A value of 1 indicates distribution, 2 indicates feeder, and 5 indicates
157     interoffice. This allows the program to differentiate between the types of
158     trenching.  In the case of distribution only copper cable need to be considered.
159     In the case of interoffice facilities only fiber cable need to be considered.
160     While the case of feeder, both copper and fiber cable need to be addressed.
161
162     Calculation for the trench investment is generally as follows (per foot):
163        Rate(s) used by routine is an average of the rate(s) found in the placement
164        file plus the shared rate(s).
165        Shared rate(s) is the placement file rate(s) divided by the number of users.
166        The weights used to calculate the average are based on the user input percent
167        sharing of buried cable.
168        If multiple rates are used the expression is compressed using the following
169        relationship:
170
171         uncompressed:          A * (1 - %shared) + (A/users) * %shared
172
173         compressed:            A * (1 - %shared + (%shared/users))
174
175        where A = sum of rates or individual rate
176   }
177
178     Case ArcType of
179     2..4 : begin  // Copper cables
180               if (ArcType = 4) or (OSPType = 'D')
181                 then begin
182                        if (BedRock > 29) and (Asmp.TrenchUsers <=2) and CanPlow
183                          {Depth of bedrock is at least 30" and TrenchUsers are 2 or less
184                          and plowing is allowed.}
185                          then begin   // using uncompressed calculation described above
186                                 if Length < 1000
187                                 then Trench_inv := ( f_OSPLabor.Labor('LP51A', Tier_Flag) * (1.0 -
188   Sharing_Buried) +
189                                                      f_OSPLabor.Labor('LP51A', Tier_Flag) * Sharing_Buried/
190   Asmp.TrenchUsers +
191                                                      f_OSPLabor.Labor('LP70C', Tier_Flag) * Sharing_Buried/
192   Asmp.TrenchUsers ) *
```

```
193                                         Length
194
195                                 else Trench_inv := ( f_OSPLabor.Labor('LP51B', Tier_Flag) * (1.0 -
196   Sharing_Buried) +
197                                         f_OSPLabor.Labor('LP51B', Tier_Flag) * Sharing_Buried/
198   Asmp.TrenchUsers +
199                                         f_OSPLabor.Labor('LP70C', Tier_Flag) * Sharing_Buried/
200   Asmp.TrenchUsers ) *
201                                         Length;
202                                  Size  := -30;    // depth of plowing required
203                                end     // END plowing allowed
204                           else begin
205                                  {Plowing not allowed; trenching investment based on
206                                  trenching with random separation in the distribution. Using
207                                  compressed calculation (with multiple rates) described above}
208                                  Length := Length * (1.0-Asmp.Percent_Trench);
209                                  if BedRock > 29   //Depth of bedrock is at least 30"
210                                    then if Length < 1000
211                                           then Trench_inv := ( f_OSPLabor.Labor('LP54A', Tier_Flag) *
212                                                                          (1.0-Per_HD-Per_Bor) +
213                                                                f_OSPLabor.Labor('LP57A', Tier_Flag) *  Per_HD
214   +
215                                                                f_OSPLabor.Labor('LP59A', Tier_Flag) *  Per_Bor
216   +
217                                                               (f_OSPLabor.Labor('LP93C', Tier_Flag)+
218                                                                f_OSPMaterial.Mat('CONCRETE',1) +
219                                                                f_OSPLabor.Labor('LP93D', Tier_Flag)) * Per_Con
220   ) *
221                                                               Length * (1.0 - Sharing_Buried + Sharing_Buried
222   /
223                                                                        Asmp.TrenchUsers)
224
225                                         else Trench_inv := ( f_OSPLabor.Labor('LP54B', Tier_Flag) *
226                                                                          (1.0 - Per_HD - Per_Bor) +
227                                                                f_OSPLabor.Labor('LP57A', Tier_Flag) *  Per_HD
228   +
229                                                                f_OSPLabor.Labor('LP59A', Tier_Flag) *  Per_Bor
230   +
231                                                               (f_OSPLabor.Labor('LP93C', Tier_Flag)+
```

```
                                                    f_OSPMaterial.Mat('CONCRETE',1) +
                                                    f_OSPLabor.Labor('LP93D', Tier_Flag)) * Per_Con
) *
                                                  Length * (1.0 - Sharing_Buried + Sharing_Buried
/
                                                           Asmp.TrenchUsers);

                                 if BedRock < 30   //Depth of bedrock is not at least 30"
                                   then if Length < 1000
                                          then Trench_inv := ( f_OSPLabor.Labor('LP54A', Tier_Flag) *
                                                                 (1.0 - Per_HD - Per_Bor) +
                                                    f_OSPLabor.Labor('LP57A', Tier_Flag) *  Per_HD
+
                                                    f_OSPLabor.Labor('LP59A', Tier_Flag) *  Per_Bor
+
                                                   (f_OSPLabor.Labor('LP93C', Tier_Flag)+
                                                    f_OSPMaterial.Mat('CONCRETE',1) +
                                                    f_OSPLabor.Labor('LP93D', Tier_Flag)) * Per_Con
+
                                                    f_OSPLabor.Labor('LP70B', Tier_Flag) ) *
                                                   Length * (1.0 - Sharing_Buried + Sharing_Buried
/
                                                           Asmp.TrenchUsers)
                                            //modified else portion 4/7/98 yv
                                        else Trench_inv := ( f_OSPLabor.Labor('LP54B', Tier_Flag) *
                                                               (1.0 - Per_HD - Per_Bor) +
                                                    f_OSPLabor.Labor('LP57A', Tier_Flag) *  Per_HD
+
                                                    f_OSPLabor.Labor('LP59A', Tier_Flag) *  Per_Bor
+
                                                   (f_OSPLabor.Labor('LP93C', Tier_Flag)+
                                                    f_OSPMaterial.Mat('CONCRETE',1) +
                                                    f_OSPLabor.Labor('LP93D', Tier_Flag)) * Per_Con
+
                                                    f_OSPLabor.Labor('LP70B', Tier_Flag) ) *
                                                   Length * (1.0 - Sharing_Buried + Sharing_Buried
/
                                                           Asmp.TrenchUsers);
                                 Size  := 30;        // depth of trench required
```

```
271                         end;    // END plowing not allowed
272                   end; //(ArcType = 4) or (OSPType = 'D')
273
274             if OSPType = 'F'
275               then begin
276                      if (BedRock > 29) and CanPlow
277                        then begin     // bedrock at least 30" deep
278                               if Length < 1000
279                                 then Trench_inv := f_OSPLabor.Labor('LP51A', Tier_Flag) * Length
280                                 else Trench_inv := f_OSPLabor.Labor('LP51B', Tier_Flag) * Length;
281                               Size  := -30;     // depth of plowing required in inches
282                             end           // END > 30"
283                        else begin
284                               {This section trenches copper cable at 30" deep.  The bedrock
285                               must be at least 42" deep because of the extra 12" of depth
286                               required to maintain 12" separation from the power cable in shared
287                               trenches.  Using uncompressed investment calculation (with multiple
288                               rates) described above.}
289                               if BedRock > 42
290                                 then begin      // bedrock >42" but <60"
291                                  if Length < 1000
292                                    then Trench_inv := ( f_OSPLabor.Labor('LP54A', Tier_Flag) *
293                                                                    (1.0 - Per_HD - Per_Bor) +
294                                                          f_OSPLabor.Labor('LP57A', Tier_Flag) *  Per_HD +
295                                                         f_OSPLabor.Labor('LP59A', Tier_Flag) *  Per_Bor +
296                                                        (f_OSPLabor.Labor('LP93C', Tier_Flag) +
297                                                         f_OSPMaterial.Mat('CONCRETE',1) +
298                                                         f_OSPLabor.Labor('LP93D', Tier_Flag)) * Per_Con ) *
299                                                         Length * (1.0-Sharing_Buried) +
300                                                       ( f_OSPLabor.Labor('LP54A', Tier_Flag) *
301                                                                    (1.0 - Per_HD - Per_Bor) +
302                                                         f_OSPLabor.Labor('LP57A', Tier_Flag) *  Per_HD +
303                                                         f_OSPLabor.Labor('LP59A', Tier_Flag) *  Per_Bor +
304                                                         f_OSPLabor.Labor('LP54C', Tier_Flag) * 2.0 +
305                                                        (f_OSPLabor.Labor('LP93C', Tier_Flag) +
306                                                         f_OSPMaterial.Mat('CONCRETE',1) +
307                                                         f_OSPLabor.Labor('LP93D', Tier_Flag))* Per_Con ) /
308                                                         Asmp.TrenchUsers * Length * Sharing_Buried
309                                  else Trench_inv := ( f_OSPLabor.Labor('LP54B', Tier_Flag) *
```

```
                                                    (1.0 - Per_HD - Per_Bor) +
                                                f_OSPLabor.Labor('LP57A', Tier_Flag) *  Per_HD +
                                                f_OSPLabor.Labor('LP59A', Tier_Flag) *  Per_Bor +
                                               (f_OSPLabor.Labor('LP93C', Tier_Flag) +
                                                f_OSPMaterial.Mat('CONCRETE',1) +
                                                f_OSPLabor.Labor('LP93D', Tier_Flag))* Per_Con ) *
                                                Length * (1.0-Sharing_Buried) +
                                              ( f_OSPLabor.Labor('LP54B', Tier_Flag) *
                                                          (1.0 - Per_HD - Per_Bor) +
                                                f_OSPLabor.Labor('LP57A', Tier_Flag) *  Per_HD +
                                                f_OSPLabor.Labor('LP54C', Tier_Flag) * 2.0 +
                                                f_OSPLabor.Labor('LP59A', Tier_Flag) *  Per_Bor +
                                               (f_OSPLabor.Labor('LP93C', Tier_Flag) +
                                                f_OSPMaterial.Mat('CONCRETE',1) +
                                                f_OSPLabor.Labor('LP93D', Tier_Flag)) * Per_Con ) /
                                                Asmp.TrenchUsers * Length * Sharing_Buried;
    Size  := 42;   // depth trench required in inches
  end        // END 42" > < 60"
  else begin  // (1) bedrock <42"
   {Copper cable is trenched at 30" deep if the bedrock is at least
   30" deep.  If the bedrock is not 42" deep for the extra 12" of
   separation required from the power cable, rock sawing is added to
   the shared portion of the trench.  Using uncompressed investment
   calculation (with multiple rates) described above.}
   if BedRock > 29
    then begin    // bedrock >30" but <42"
      if Length < 1000
       then Trench_inv := ( f_OSPLabor.Labor('LP54A', Tier_Flag) *
                                          (1.0 - Per_HD - Per_Bor) +
                            f_OSPLabor.Labor('LP57A', Tier_Flag) *  Per_HD +
                            f_OSPLabor.Labor('LP59A', Tier_Flag) *  Per_Bor +
                           (f_OSPLabor.Labor('LP93C', Tier_Flag) +
                            f_OSPMaterial.Mat('CONCRETE',1) +
                            f_OSPLabor.Labor('LP93D', Tier_Flag)) * Per_Con )
*
                            Length * (1.0-Sharing_Buried) +
                          ( f_OSPLabor.Labor('LP54A', Tier_Flag) *
                                      (1.0 - Per_HD - Per_Bor) +
                            f_OSPLabor.Labor('LP57A', Tier_Flag) *  Per_HD +
```

```
                                                f_OSPLabor.Labor('LP54C', Tier_Flag) * 2.0 +
                                                f_OSPLabor.Labor('LP70B', Tier_Flag) +
                                                f_OSPLabor.Labor('LP59A', Tier_Flag) *  Per_Bor +
                                               (f_OSPLabor.Labor('LP93C', Tier_Flag) +
                                                f_OSPMaterial.Mat('CONCRETE',1) +
                                                f_OSPLabor.Labor('LP93D', Tier_Flag))* Per_Con )
/
                                                Asmp.TrenchUsers * Length * Sharing_Buried
                                  else Trench_inv := ( f_OSPLabor.Labor('LP54B', Tier_Flag) *
                                                              (1.0 - Per_HD - Per_Bor) +
                                                f_OSPLabor.Labor('LP57A', Tier_Flag) *  Per_HD +
                                                f_OSPLabor.Labor('LP59A', Tier_Flag) *  Per_Bor +
                                               (f_OSPLabor.Labor('LP93C', Tier_Flag) +
                                                f_OSPMaterial.Mat('CONCRETE',1) +
                                                f_OSPLabor.Labor('LP93D', Tier_Flag)) * Per_Con )
*
                                                Length * (1.0-Sharing_Buried) +
                                              ( f_OSPLabor.Labor('LP54B', Tier_Flag) *
                                                              (1.0 - Per_HD - Per_Bor) +
                                                f_OSPLabor.Labor('LP57A', Tier_Flag) *  Per_HD +
                                                f_OSPLabor.Labor('LP54C', Tier_Flag) * 2.0 +
                                                f_OSPLabor.Labor('LP70B', Tier_Flag) +
                                                f_OSPLabor.Labor('LP59A', Tier_Flag) *  Per_Bor +
                                               (f_OSPLabor.Labor('LP93C', Tier_Flag) +
                                                f_OSPMaterial.Mat('CONCRETE',1) +
                                                f_OSPLabor.Labor('LP93D', Tier_Flag))* Per_Con )
/
                                                Asmp.TrenchUsers * Length * Sharing_Buried;
                                     Size  := 42;     // depth trench required in inches
                                  end   // END >30" but <42"

                                else begin    // (2) bedrock <42"
                                     {Copper cable is trenched at 30" deep.  If the bedrock is not at
                                     least 30" deep, rock sawing is added to BOTH the non-shared and
                                     shared portions of the trench.  Using uncompressed investment
                                     calculation (with multiple rates) described above.}
                                     if BedRock < 30
                                      then begin         // bedrock <30"
                                      if Length < 1000
```

```
388                                          then Trench_inv := ( f_OSPLabor.Labor('LP54A', Tier_Flag) *
389                                                                 (1.0 - Per_HD - Per_Bor) +
390                                                               f_OSPLabor.Labor('LP57A', Tier_Flag) *
391   Per_HD +
392                                                               f_OSPLabor.Labor('LP70B', Tier_Flag) +
393                                                               f_OSPLabor.Labor('LP59A', Tier_Flag) *
394   Per_Bor +
395                                                              (f_OSPLabor.Labor('LP93C', Tier_Flag) +
396                                                               f_OSPMaterial.Mat('CONCRETE',1) +
397                                                               f_OSPLabor.Labor('LP93D', Tier_Flag))*
398   Per_Con ) *
399                                                               Length * (1.0-Sharing_Buried) +
400                                                             ( f_OSPLabor.Labor('LP54A', Tier_Flag) *
401                                                                 (1.0 - Per_HD - Per_Bor) +
402                                                               f_OSPLabor.Labor('LP57A', Tier_Flag) *
403   Per_HD +
404                                                               f_OSPLabor.Labor('LP54C', Tier_Flag) * 2.0 +
405                                                               f_OSPLabor.Labor('LP70B', Tier_Flag) +
406                                                               f_OSPLabor.Labor('LP59A', Tier_Flag) *
407   Per_Bor +
408                                                              (f_OSPLabor.Labor('LP93C', Tier_Flag) +
409                                                               f_OSPMaterial.Mat('CONCRETE',1) +
410                                                               f_OSPLabor.Labor('LP93D', Tier_Flag))*
411   Per_Con ) /
412                                                               Asmp.TrenchUsers * Length * Sharing_Buried
413                                          else Trench_inv := ( f_OSPLabor.Labor('LP54B', Tier_Flag) *
414                                                                 (1.0 - Per_HD - Per_Bor) +
415                                                               f_OSPLabor.Labor('LP57A', Tier_Flag) *
416   Per_HD +
417                                                               f_OSPLabor.Labor('LP70B', Tier_Flag) +
418                                                               f_OSPLabor.Labor('LP59A', Tier_Flag) *
419   Per_Bor +
420                                                              (f_OSPLabor.Labor('LP93C', Tier_Flag) +
421                                                               f_OSPMaterial.Mat('CONCRETE',1) +
422                                                               f_OSPLabor.Labor('LP93D', Tier_Flag))*
423   Per_Con ) *
424                                                               Length * (1.0-Sharing_Buried) +
425                                                             ( f_OSPLabor.Labor('LP54B', Tier_Flag) *
426                                                                 (1.0 - Per_HD - Per_Bor) +
```

```
427                                                                    f_OSPLabor.Labor('LP57A', Tier_Flag) *
428   Per_HD +
429                                                                    f_OSPLabor.Labor('LP54C', Tier_Flag) * 2.0 +
430                                                                    f_OSPLabor.Labor('LP70B', Tier_Flag) +
431                                                                    f_OSPLabor.Labor('LP59A', Tier_Flag) *
432   Per_Bor +
433                                                                   (f_OSPLabor.Labor('LP93C', Tier_Flag) +
434                                                                    f_OSPMaterial.Mat('CONCRETE',1) +
435                                                                    f_OSPLabor.Labor('LP93D', Tier_Flag))*
436   Per_Con ) /
437                                                                    Asmp.TrenchUsers * Length * Sharing_Buried;
438                                            Size  := 30;    // depth trench required in inches
439                                          end;          // END <30"
440                               end;    // END (2) <42"
441                             end;     // END (1) <42"
442                           end;
443                    end;
444
445          end;      // END Copper Cable
446
447     { This section calculates the investments for plowing feeder and transport cables.
448       Fiber cables are either plowed at 48" or at 30" in subduct for additional protection.}
449     0,1 : begin    // Fiber Cable
450             if CanPlow
451              then begin     // plowing is allowed when density < 50 lines per square mile
452                     if BedRock > 47
453                      then begin       // bedrock >= 48" deep
454                            if Length < 1000
455                             then Trench_inv := ( f_OSPLabor.Labor('LP51A', Tier_Flag) +
456                                                  f_OSPLabor.Labor('LP51C', Tier_Flag) * 3.0 +
457                                                  f_OSPLabor.Labor('LP52A', Tier_Flag) * 0.10 ) * Length
458                             else Trench_inv := ( f_OSPLabor.Labor('LP51B', Tier_Flag) +
459                                                  f_OSPLabor.Labor('LP51C', Tier_Flag) * 3.0 +
460                                                  f_OSPLabor.Labor('LP52A', Tier_Flag) * 0.10 ) * Length;
461                             Size  := -48;    // depth of plowing required in inches
462                            end    // END > 48"
463
464                      else if (BedRock < 49) and (BedRock > 29)
465                              then begin       // bedrock between 30" & 48" deep
```

```
                                    // fiber cable placed in the subduct for protection
                                    if Length < 1000
                                     then Trench_inv := ( f_OSPLabor.Labor('LP51A', Tier_Flag) +
                                                          f_OSPLabor.Labor('LP52A', Tier_Flag) * 0.10 +
                                                          f_OSPLabor.Labor('LP43C', Tier_Flag) +
                                                          f_OSPMaterial.Mat('SUBDUC', 1) ) * Length
                                     else Trench_inv := ( f_OSPLabor.Labor('LP51B', Tier_Flag) +
                                                          f_OSPLabor.Labor('LP52A', Tier_Flag) * 0.10 +
                                                          f_OSPLabor.Labor('LP43C', Tier_Flag) +
                                                          f_OSPMaterial.Mat('SUBDUC', 1) ) * Length;
                                      Size  := -30;       // depth of plowing required in inches
                                    end;   // END 30" > < 48"

              end    // END Can plow

       else begin    // Cannot plow when bedrock not at least 30" deep, must use trenching
             //This section trenches fiber cable at 48" deep.  The bedrock
             //must be at least 60" deep because of the extra 12" of depth
             //required to maintain 12" separation from the power cable in
             //shared trenches.  Using uncompressed investment calculation
             //(with multiple rates) described above.

             if BedRock > 59
              then begin   // bedrock at least 60" deep
              if Length < 1000
               then Trench_inv := ( f_OSPLabor.Labor('LP54A', Tier_Flag) * (1.0 - Per_HD - Per_Bor) +
                                    f_OSPLabor.Labor('LP57A', Tier_Flag) *  Per_HD +
                                    f_OSPLabor.Labor('LP54C', Tier_Flag) * 3.0 +
                                    f_OSPLabor.Labor('LP59A', Tier_Flag) *  Per_Bor +
                                   (f_OSPLabor.Labor('LP93C', Tier_Flag)+
                                    f_OSPMaterial.Mat('CONCRETE',1) +
                                    f_OSPLabor.Labor('LP93D', Tier_Flag)) * Per_Con ) *
                                    Length * (1.0-Sharing_Buried) +
                                  ( f_OSPLabor.Labor('LP54A', Tier_Flag) * (1.0 - Per_HD - Per_Bor) +
                                    f_OSPLabor.Labor('LP57A', Tier_Flag) *  Per_HD +
                                    f_OSPLabor.Labor('LP54C', Tier_Flag) * 5.0 +
                                    f_OSPLabor.Labor('LP59A', Tier_Flag) *  Per_Bor +
                                   (f_OSPLabor.Labor('LP93C', Tier_Flag)+
                                    f_OSPMaterial.Mat('CONCRETE',1) +
```

```
505                                f_OSPLabor.Labor('LP93D', Tier_Flag)) * Per_Con ) /
506                                Asmp.TrenchUsers * Length * Sharing_Buried
507                    else Trench_inv := ( f_OSPLabor.Labor('LP54B', Tier_Flag) * (1.0 - Per_HD - Per_Bor) +
508                                f_OSPLabor.Labor('LP57A', Tier_Flag) *  Per_HD +
509                                f_OSPLabor.Labor('LP54C', Tier_Flag) * 3.0 +
510                                f_OSPLabor.Labor('LP59A', Tier_Flag) *  Per_Bor +
511                               (f_OSPLabor.Labor('LP93C', Tier_Flag)+
512                                f_OSPMaterial.Mat('CONCRETE',1) +
513                                f_OSPLabor.Labor('LP93D', Tier_Flag)) * Per_Con ) *
514                                Length * (1.0-Sharing_Buried) +
515                              ( f_OSPLabor.Labor('LP54B', Tier_Flag) * (1.0 - Per_HD - Per_Bor) +
516                                f_OSPLabor.Labor('LP57A', Tier_Flag) *  Per_HD +
517                                f_OSPLabor.Labor('LP54C', Tier_Flag) * 5.0 +
518                                f_OSPLabor.Labor('LP59A', Tier_Flag) *  Per_Bor +
519                               (f_OSPLabor.Labor('LP93C', Tier_Flag)+
520                                f_OSPMaterial.Mat('CONCRETE',1) +
521                                f_OSPLabor.Labor('LP93D', Tier_Flag)) * Per_Con ) /
522                                Asmp.TrenchUsers * Length * Sharing_Buried;
523                     Size  := 60;   // depth of trench required in inches
524                    end    // END 60"
525
526                    else begin        // (1)bedrock <60" deep
527                    {Fiber cable is trenched at 30" deep if the bedrock is not 60"
528                    deep but is at least 42" deep for the extra 12" of separation
529                    required from the power cable.  Subduct is placed at 30" in the
530                    trench and the fiber placed in the subduct for protection.  Using
531                    uncompressed investment calculation (with multiple rates) described
532                    above.}
533                    if BedRock > 42
534                     then begin       // bedrock >42" but <60" deep
535                     if Length < 1000
536                      then Trench_inv := ( f_OSPLabor.Labor('LP54A', Tier_Flag) * (1.0 - Per_HD - Per_Bor) +
537                                           f_OSPLabor.Labor('LP57A', Tier_Flag) *  Per_HD +
538                                           f_OSPLabor.Labor('LP43C', Tier_Flag) +
539                                           f_OSPMaterial.Mat('SUBDUC', 1)        +
540                                           f_OSPLabor.Labor('LP59A', Tier_Flag) *  Per_Bor +
541                                          (f_OSPLabor.Labor('LP93C', Tier_Flag) +
542                                           f_OSPMaterial.Mat('CONCRETE',1) +
543                                           f_OSPLabor.Labor('LP93D', Tier_Flag)) * Per_Con ) *
```

```
                                    Length * (1.0-Sharing_Buried) +
                                  ( f_OSPLabor.Labor('LP54A', Tier_Flag) * (1.0 - Per_HD - Per_Bor) +
                                    f_OSPLabor.Labor('LP57A', Tier_Flag) *  Per_HD +
                                    f_OSPLabor.Labor('LP54C', Tier_Flag) * 2.0 +
                                    f_OSPLabor.Labor('LP43C', Tier_Flag) +
                                    f_OSPMaterial.Mat('SUBDUC', 1)        +
                                    f_OSPLabor.Labor('LP59A', Tier_Flag) *  Per_Bor +
                                   (f_OSPLabor.Labor('LP93C', Tier_Flag) +
                                    f_OSPMaterial.Mat('CONCRETE',1) +
                                    f_OSPLabor.Labor('LP93D', Tier_Flag))* Per_Con ) /
                                    ASmp.TrenchUsers * Length * Sharing_Buried
        else Trench_inv := ( f_OSPLabor.Labor('LP54B', Tier_Flag) * (1.0 - Per_HD - Per_Bor) +
                                    f_OSPLabor.Labor('LP57A', Tier_Flag) *  Per_HD +
                                    f_OSPLabor.Labor('LP43C', Tier_Flag) +
                                    f_OSPMaterial.Mat('SUBDUC', 1)        +
                                    f_OSPLabor.Labor('LP59A', Tier_Flag) *  Per_Bor +
                                   (f_OSPLabor.Labor('LP93C', Tier_Flag) +
                                    f_OSPMaterial.Mat('CONCRETE',1) +
                                    f_OSPLabor.Labor('LP93D', Tier_Flag))* Per_Con ) *
                                    Length * (1.0-Sharing_Buried) +
                                  ( f_OSPLabor.Labor('LP54B', Tier_Flag) * (1.0 - Per_HD - Per_Bor) +
                                    f_OSPLabor.Labor('LP57A', Tier_Flag) *  Per_HD +
                                    f_OSPLabor.Labor('LP54C', Tier_Flag) * 2.0 +
                                    f_OSPLabor.Labor('LP43C', Tier_Flag) +
                                    f_OSPMaterial.Mat('SUBDUC', 1)        +
                                    f_OSPLabor.Labor('LP59A', Tier_Flag) *  Per_Bor +
                                   (f_OSPLabor.Labor('LP93C', Tier_Flag) +
                                    f_OSPMaterial.Mat('CONCRETE',1) +
                                    f_OSPLabor.Labor('LP93D', Tier_Flag)) * Per_Con ) /
                                    Asmp.TrenchUsers * Length * Sharing_Buried;
          Size  := 42;    // depth of trench required in inches
        end                // END 42" > < 60"
        else begin        // (2)bedrock <60" deep
        {Fiber cable is trenched at 30" deep if the bedrock is at least
        30" deep. If the bedrock is not 42" deep for the extra 12" of
        separation required from the power cable, rock sawing is added.
        The fiber is placed in the subduct for protection. Using
        uncompressed investment calculation (with multiple rates)
        described above.}
```

```
583                     if BedRock > 29
584                      then begin      // bedrock >30" but <42"
585                      if Length < 1000
586                       then Trench_inv := ( f_OSPLabor.Labor('LP54A', Tier_Flag) * (1.0 - Per_HD - Per_Bor) +
587                                            f_OSPLabor.Labor('LP57A', Tier_Flag) *  Per_HD +
588                                            f_OSPLabor.Labor('LP43C', Tier_Flag) +
589                                            f_OSPMaterial.Mat('SUBDUC', 1)        +
590                                            f_OSPLabor.Labor('LP59A', Tier_Flag) *  Per_Bor +
591                                           (f_OSPLabor.Labor('LP93C', Tier_Flag) +
592                                            f_OSPMaterial.Mat('CONCRETE',1) +
593                                            f_OSPLabor.Labor('LP93D', Tier_Flag)) * Per_Con ) *
594                                            Length • (1.0-Sharing_Buried) +
595                                          ( f_OSPLabor.Labor('LP54A', Tier_Flag) * (1.0 - Per_HD - Per_Bor) +
596                                            f_OSPLabor.Labor('LP57A', Tier_Flag) *  Per_HD +
597                                            f_OSPLabor.Labor('LP54C', Tier_Flag) * 2.0 +
598                                            f_OSPLabor.Labor('LP70B', Tier_Flag) +
599                                            f_OSPLabor.Labor('LP43C', Tier_Flag) +
600                                            f_OSPMaterial.Mat('SUBDUC', 1)        +
601                                            f_OSPLabor.Labor('LP59A', Tier_Flag) *  Per_Bor +
602                                           (f_OSPLabor.Labor('LP93C', Tier_Flag) +
603                                            f_OSPMaterial.Mat('CONCRETE',1) +
604                                            f_OSPLabor.Labor('LP93D', Tier_Flag))* Per_Con ) /
605                                            Asmp.TrenchUsers * Length * Sharing_Buried
606                       else Trench_inv := ( f_OSPLabor.Labor('LP54B', Tier_Flag) * (1.0 - Per_HD - Per_Bor) +
607                                            f_OSPLabor.Labor('LP57A', Tier_Flag) *  Per_HD +
608                                            f_OSPLabor.Labor('LP43C', Tier_Flag) +
609                                            f_OSPMaterial.Mat('SUBDUC', 1)        +
610                                            f_OSPLabor.Labor('LP59A', Tier_Flag) *  Per_Bor +
611                                           (f_OSPLabor.Labor('LP93C', Tier_Flag) +
612                                            f_OSPMaterial.Mat('CONCRETE',1) +
613                                            f_OSPLabor.Labor('LP93D', Tier_Flag)) * Per_Con ) *
614                                            Length * (1.0-Sharing_Buried) +
615                                          ( f_OSPLabor.Labor('LP54B', Tier_Flag) * (1.0 - Per_HD - Per_Bor) +
616                                            f_OSPLabor.Labor('LP57A', Tier_Flag) *  Per_HD +
617                                            f_OSPLabor.Labor('LP54C', Tier_Flag) * 2.0 +
618                                            f_OSPLabor.Labor('LP70B', Tier_Flag) +
619                                            f_OSPLabor.Labor('LP43C', Tier_Flag) +
620                                            f_OSPMaterial.Mat('SUBDUC', 1)        +
621                                            f_OSPLabor.Labor('LP59A', Tier_Flag) *  Per_Bor +
```

```
                                        (f_OSPLabor.Labor('LP93C', Tier_Flag) +
                                         f_OSPMaterial.Mat('CONCRETE',1) +
                                         f_OSPLabor.Labor('LP93D', Tier_Flag))* Per_Con ) /
                                         Asmp.TrenchUsers * Length * Sharing_Buried;
                  Size  := 42;      // depth of trench required in inches
                end                  // END >30" but <42"

                else begin          // (3)bedrock <60" deep
                {Fiber cable is trenched at 30" deep. If the bedrock is not at
                least 30" deep, rock sawing is added.  The fiber is placed in
                the subduct for protection.  Using uncompressed investment
                calculation (with multiple rates) described above.}
                if BedRock < 30
                 then begin          // bedrock <30"
                 if Length < 1000
                  then Trench_inv := ( f_OSPLabor.Labor('LP54A', Tier_Flag) * (1.0 - Per_HD - Per_Bor)
+
                                        f_OSPLabor.Labor('LP57A', Tier_Flag) *  Per_HD +
                                        f_OSPLabor.Labor('LP43C', Tier_Flag) +
                                        f_OSPLabor.Labor('LP70B', Tier_Flag) +
                                        f_OSPMaterial.Mat('SUBDUC', 1)        +
                                        f_OSPLabor.Labor('LP59A', Tier_Flag) *  Per_Bor +
                                       (f_OSPLabor.Labor('LP93C', Tier_Flag) +
                                        f_OSPMaterial.Mat('CONCRETE',1) +
                                        f_OSPLabor.Labor('LP93D', Tier_Flag))* Per_Con ) *
                                        Length * (1.0-Sharing_Buried) +
                                       ( f_OSPLabor.Labor('LP54A', Tier_Flag) * (1.0 - Per_HD - Per_Bor)
+
                                        f_OSPLabor.Labor('LP57A', Tier_Flag) *  Per_HD +
                                        f_OSPLabor.Labor('LP54C', Tier_Flag) * 2.0 +
                                        f_OSPLabor.Labor('LP70B', Tier_Flag) +
                                        f_OSPLabor.Labor('LP43C', Tier_Flag) +
                                        f_OSPMaterial.Mat('SUBDUC', 1)        +
                                        f_OSPLabor.Labor('LP59A', Tier_Flag) *  Per_Bor +
                                       (f_OSPLabor.Labor('LP93C', Tier_Flag) +
                                        f_OSPMaterial.Mat('CONCRETE',1) +
                                        f_OSPLabor.Labor('LP93D', Tier_Flag))* Per_Con ) /
                                        Asmp.TrenchUsers * Length * Sharing_Buried
```

```
660                        else Trench_inv := ( f_OSPLabor.Labor('LP54B', Tier_Flag) * (1.0 - Per_HD - Per_Bor)
661   +
662                                             f_OSPLabor.Labor('LP57A', Tier_Flag) *  Per_HD +
663                                             f_OSPLabor.Labor('LP43C', Tier_Flag) +
664                                             f_OSPLabor.Labor('LP70B', Tier_Flag) +
665                                             f_OSPMaterial.Mat('SUBDUC', 1)        +
666                                             f_OSPLabor.Labor('LP59A', Tier_Flag) *  Per_Bor +
667                                            (f_OSPLabor.Labor('LP93C', Tier_Flag) +
668                                             f_OSPMaterial.Mat('CONCRETE',1) +
669                                             f_OSPLabor.Labor('LP93D', Tier_Flag))* Per_Con ) *
670                                             Length * (1.0-Sharing_Buried) +
671                                           ( f_OSPLabor.Labor('LP54B', Tier_Flag) * (1.0 - Per_HD - Per_Bor)
672   +
673                                             f_OSPLabor.Labor('LP57A', Tier_Flag) *  Per_HD +
674                                             f_OSPLabor.Labor('LP54C', Tier_Flag) * 2.0 +
675                                             f_OSPLabor.Labor('LP70B', Tier_Flag) +
676                                             f_OSPLabor.Labor('LP43C', Tier_Flag) +
677                                             f_OSPMaterial.Mat('SUBDUC', 1)        +
678                                             f_OSPLabor.Labor('LP59A', Tier_Flag) *  Per_Bor +
679                                            (f_OSPLabor.Labor('LP93C', Tier_Flag) +
680                                             f_OSPMaterial.Mat('CONCRETE',1) +
681                                             f_OSPLabor.Labor('LP93D', Tier_Flag))* Per_Con ) /
682                                             Asmp.TrenchUsers * Length * Sharing_Buried;
683                           Size  := 42;    // depth of trench required in inches
684                         end      // END <30"
685                        end;   // END (3)bedrock <60" deep
686                      end;  // END (2)bedrock <60" deep
687                    end; // END (1)bedrock <60" deep (
688
689               end; // END Can not plow
690            end; // END Case 2: Feeder & Transport
691       end;
692     //-----------------------------------------
693        {write trench information to the Inventory File }
694        if Size < 0
695         then PartInventory.Write (Clli, OSPType[1], 'BURPLCP', -Size, round(Length), Xpos, Ypos)  // Plowed
696         else PartInventory.Write (Clli, OSPType[1], 'BURPLCT',  Size, round(Length), Xpos, Ypos); // Trenched
697        {return the trench investment value }
698        Trench := Trench_inv;
```

```
699   end;        // END of Trench function
700
701   function Conduit(Clli : String; ArcType, Xpos, Ypos, Demand, Bus_Lines : integer;
702                            Length : Double; InvType: String) : Double;
703
704  { The Conduit function calculates the investment for conduit, manholes, and pull
705    boxes required to support the underground cable segment.  The "CalcCon" function
706    which does all the investment calculations, is embedded inside the "Conduit"
707    function.
708
709       Labor Activity Codes:  (used in Conduit investment calculation)
710          LP49B:      Place Cable/Duct in open Trench (all types/size)
711          LP54A:      Trench - Short Distance (up to 1000' - 30" depth)
712          LP55A:      Backhoe Buried Cable (up to 36" depth)
713          LP55B:      Backhoe - Additional Depth (each additional 12" depth increment)
714          LP70A:      Solid Rock
715          LP70B:      Rock Saw
716          LP73B:      Well Point
717          LP87A:      Rod and Mandrel Duct
718          LP88A:      Place Sub-duct or Air Tube in Conduit
719          LP93C:      Cut and Remove Concrete (0 to 100 square feet)
720          LP93D:      Place Concrete  (0 to 100 square feet)
721        MNHOLE:         Manhole install
722          PULLBOX:    Pull Box install
723   }
724
725   Var
726     Size      : Integer;      // Number of conduits
727     Norm_Inv  : Double;         // non-shared conduit investment
728     Share_Inv : Double;         // shared conduit investment
729
730   function CalcCon : Double;    // calculates the conduit investment
731     Var
732      Duct_Inv     : Double;     // conduit investment
733      TrenchAdd    : Double;     // additional trench depth depending on number of ducts places
734      Vs_Conduit   : Double;     // volume sensitive conduit investment
735      Man_Pull_Cnt : Longint;    // number of manholes (if UseManHole=True) or pull boxes
736      Per_Con      : Double;     // percent concrete
737
```

```
738     begin    // beginning of function CalcCon code
739       {Determines the percent of concrete that will be applied in the trenching
740       for conduit, based on Structure_Flag (0=low density, 1=med to high density,
741       2=high density business, distribution only).}
742       Case Asmp.Structure_Flag  of
743        0: Per_Con := 0.0;
744        1: if PercntConcrete < 0.0
745             then Per_Con := 0.0
746             else Per_Con := Asmp.Percent_Concrete;
747        2: Per_Con := 0.80;
748       end;
749
750       {calculate the conduit investment (duct_inv) - sum material cost and labor
751       for placed additional ducts; placing the first duct is included in the
752       trenching labor }
753       duct_inv := Length * (f_OSPMaterial.Mat('COND', Size)+
754                   f_OSPLabor.Labor('LP49B',Tier_Flag)*(Size-1));
755
756       {write conduit information to Inventory File }
757       if InvType = 'AVG'
758         then Case ArcType of
759                0 : PartInventory.Write (Clli, 'I', 'CONDUIT', Size, Round(Length), XPos, YPos);
760                1 : PartInventory.Write (Clli, 'F', 'CONDUIT', Size, Round(Length), XPos, YPos);
761              2,3 : PartInventory.Write (Clli, OSPType[1], 'CONDUIT', Size, Round(Length), XPos, YPos);
762                4 : PartInventory.Write (Clli, 'D', 'CONDUIT', Size, Round(Length), XPos, YPos);
763               end;
764
765       {set volume sensitive conduit investment equal to the conduit investment }
766       Vs_Conduit := duct_inv;
767
768       TrenchAdd := 0.0;
769       {Determine number of additional increments of trench depths (TrenchAdd)
770       required or if man holes (UseManHole) are needed, each depending on
771       number of ducts ('Size').  TrenchAdd is applied later to the labor
772       rate.  If UseManHole is False then pull boxes are used.}
773       if Size > 2
774         then if Size > 12
775                   then TrenchAdd := 2.0    // add 2 increments of trench depth
776                   else TrenchAdd := 1.0;   // add one increment of trench depth
```

```
777
778       {Re-calculate the trenching investment for the conduit depending on the
779       number of ducts placed.  If less than 3 ducts (2 is minimum placed) are
780       placed, trenching is assumed to be done by a trencher instead of a backhoe.
781       Investment calculated by summing weighted labor and material.}
782       if ( Size < 3)
783         then duct_inv := duct_inv + length * (f_OSPLabor.Labor('LP54A',Tier_Flag) +
784                                               (f_OSPLabor.Labor('LP93C', Tier_Flag) +
785                                                f_OSPMaterial.Mat('CONCRETE',1) +
786                                                f_OSPLabor.Labor('LP93D', Tier_Flag))* Per_Con+
787                                              f_OSPLabor.Labor('LP87A',Tier_Flag))
788         {When more than 2 ducts are placed the trenching is assumed to be done
789         by a backhoe.  The additional trench depths (TrenchAdd) are included here.
790         Investment calculated by summing weighted labor and material.}
791         else duct_inv := Duct_Inv + Length *
792                              ( f_OSPLabor.Labor('LP55A',Tier_Flag) +
793                                f_OSPLabor.Labor('LP55B',Tier_Flag)* TrenchAdd +
794                                (f_OSPLabor.Labor('LP93C', Tier_Flag) +
795                                 f_OSPMaterial.Mat('CONCRETE',1) +
796                                 f_OSPLabor.Labor('LP93D', Tier_Flag))* Per_Con +
797                                f_OSPLabor.Labor('LP87A',Tier_Flag) );
798
799       if BedRock < 36    // bedrock within 36" of surface
800         then begin
801                {add rock sawing information to the conduit investment }
802                duct_inv := duct_inv + Length * f_OSPLabor.Labor('LP70B',Tier_Flag);
803              end;  // END bedrock within 36" fo surface
804
805       if ArcType =4
806         then begin  // Distribution
807                if (Bus_Lines>60.0) or (Size >2)
808                {Manholes placed in distribution and business lines >60 }
809                  then begin    // place manholes
810                         {Determine number of manholes using cable length,
811                          road feet, and cable  sections(Nodes).  Add manhole
812                          information to conduit investment.}
813                         Man_Pull_Cnt := round(Length/Asmp.ManHole_Spacing);
814                         duct_inv     := Duct_Inv + Man_Pull_Cnt *
815                                         (f_OSPLabor.Labor('MNHOLE',Tier_Flag)+
```

```
816                                          f_OSPMaterial.Mat('MNHOLE', 1));
817
818                          if BedRock < 72
819                             then duct_inv := duct_inv + Asmp.LnFtManhole * Round(Length/Asmp.ManHole_Spacing)
820   *
821                                          f_OSPLabor.Labor('LP70A',Tier_Flag);
822
823                          if (WaterTable < 48)
824                            {water table within 48" of surface and manholes placed - add well
825                             points information to the conduit investment }
826                            then duct_inv := duct_inv + Asmp.WellPointDays *
827   Round(Length/Asmp.ManHole_Spacing) *
828                                          f_OSPLabor.Labor('LP73B',Tier_Flag);
829
830                          if (InvType = 'AVG') and (Man_Pull_Cnt > 0)
831                           {write manhole information to Inventory File }
832                             then PartInventory.Write (Clli, OSPType[1], 'MANHOLE',
833   round(Length/Man_Pull_Cnt),
834                                                    Man_Pull_Cnt, Xpos, Ypos);
835                       end  // END Bus Lines > 60
836
837                       else if (Bus_Lines>6.0)
838                             {Pullbox placed in distribution and business lines >6 }
839                               then begin // place pull boxes
840                                      {Determine number of pull boxes using cable length,
841                                       road feet, and cable sections (Nodes).  Add pull
842                                       box information to conduit investment.}
843                                      Man_Pull_Cnt := round(Length/Asmp.ManHole_Spacing);
844                                      duct_inv     := Duct_Inv + Man_Pull_Cnt *
845                                                      (f_OSPLabor.Labor('PULLBOX',Tier_Flag)+
846                                                      f_OSPMaterial.Mat('PULLBOX', 1));
847
848                                      if BedRock < 48 // Rock removal
849                                        then duct_inv := duct_inv + Asmp.LnFtPullBox *
850   Round(Length/Asmp.ManHole_Spacing) *
851                                             f_OSPLabor.Labor('LP70A',Tier_Flag);
852
853                                      if (InvType = 'AVG') and (Man_Pull_Cnt > 0)
854                                       {write pull box information to Inventory File }
```

```
855                                     then PartInventory.Write (Clli, OSPType[1], 'PULLBOX',
856   round(Length/Man_Pull_Cnt),
857                                                                  Man_Pull_Cnt,  Xpos, Ypos);
858                                  end;          // END Bus Lines > 6
859               end   // End of Distribution
860
861           else begin // Feeder or Tranport Fiber
862                  if ArcType < 2
863                    then begin   // Fiber cable placed
864                           duct_inv   := duct_inv + Size * Length * f_OSPLabor.Labor('LP88A',Tier_Flag) +
865                                         3.0 * Size * Length * f_OSPMaterial.Mat('SUBDUC', 1);
866                           Vs_Conduit := Vs_Conduit + Size * Length * f_OSPLabor.Labor('LP88A',Tier_Flag) +
867                                         3.0 * Size * Length * f_OSPMAterial.Mat('SUBDUC', 1);
868
869                           {write subduct info to Inventory File }
870                           if InvType = 'AVG' then PartInventory.Write (Clli, OSPType[1], 'SUBDUCT', 3*Size,
871   round(Length), Xpos, Ypos);
872
873                           if Size > 2
874                             then begin // Place manholes
875                                    Man_Pull_Cnt := round(Length/Asmp.PullBox_Spacing);
876                                    duct_inv := Duct_Inv + Man_Pull_Cnt *
877                                                (f_OSPLabor.Labor('MNHOLE',Tier_Flag)+
878                                                 f_OSPMaterial.Mat('MNHOLE', 1));
879
880                                    if BedRock < 72
881                                      then duct_inv := duct_inv + Asmp.LnFtManhole *
882   Round(Length/Asmp.PullBox_Spacing) *
883                                                     f_OSPLabor.Labor('LP70A',Tier_Flag);
884                                    if (WaterTable < 48)
885                                     {water table within 48" of surface and manholes placed - add well
886                                      points information to the conduit investment }
887                                       then duct_inv := duct_inv + Asmp.WellPointDays *
888   Round(Length/Asmp.PullBox_Spacing) *
889                                                      f_OSPLabor.Labor('LP73B',Tier_Flag);
890
891                                    if (InvType = 'AVG') and (Man_Pull_Cnt > 0)
892                                     {write manhole information to Inventory File }
```

```
893                                  then PartInventory.Write (Clli, OSPType[1], 'MANHOLE',
894   round(Length/Man_Pull_Cnt),
895                                                          Man_Pull_Cnt, Xpos, Ypos);
896                                end // END Place manholes
897
898                             else begin // Place pullboxes
899                                    Man_Pull_Cnt := round(Length/Asmp.PullBox_Spacing);
900                                    duct_inv := Duct_Inv + Man_Pull_Cnt *
901                                                (f_OSPLabor.Labor('PULLBOX',Tier_Flag)+
902                                                 f_OSPMaterial.Mat('PULLBOX', 1));
903
904                                    if BedRock < 48 // Rock removal
905                                      then duct_inv := duct_inv + Asmp.LnFtPullBox *
906   Round(Length/Asmp.PullBox_Spacing) *
907                                                       f_OSPLabor.Labor('LP70A',Tier_Flag);
908
909                                    if (InvType = 'AVG') and (Man_Pull_Cnt > 0)
910                                     {write manhole information to Inventory File }
911                                      then PartInventory.Write (Clli, OSPType[1], 'PULLBOX',
912   round(Length/Man_Pull_Cnt),
913                                                          Man_Pull_Cnt, Xpos, Ypos);
914                                end; // End Place pullboxes
915                        end;     // End Fiber cable placed
916
917                 if ArcType in [2,3]
918                   then begin   // Copper cable placed
919                          if Demand >= 400
920                            then begin // Place manholes
921                                   Man_Pull_Cnt := round(Length/Asmp.ManHole_Spacing);
922                                   duct_inv := Duct_Inv + Man_Pull_Cnt *
923                                               (f_OSPLabor.Labor('MNHOLE',Tier_Flag)+
924                                                f_OSPMaterial.Mat('MNHOLE', 1));
925
926                                   if BedRock < 72
927                                     then duct_inv := duct_inv + Asmp.LnFtManhole *
928   Round(Length/Asmp.ManHole_Spacing) *
929                                                       f_OSPLabor.Labor('LP70A',Tier_Flag);
930                                   if (WaterTable < 48)
931                                    {water table within 48" of surface and manholes placed - add well
```

```
932                                   points information to the conduit investment }
933                                    then duct_inv := duct_inv + Asmp.WellPointDays *
934   Round(Length/Asmp.ManHole_Spacing) *
935                                                      f_OSPLabor.Labor('LP73B',Tier_Flag);
936
937                                  if (InvType = 'AVG') and (Man_Pull_Cnt > 0)
938                                   {write manhole information to Inventory File }
939                                    then PartInventory.Write (Clli, OSPType[1], 'MANHOLE',
940   round(Length/Man_Pull_Cnt),
941                                                                Man_Pull_Cnt, Xpos, Ypos);
942                               end   // END Place manholes
943
944                           else begin // Place pullboxes
945                                  Man_Pull_Cnt := round(Length/Asmp.ManHole_Spacing);
946                                  duct_inv := Duct_Inv + Man_Pull_Cnt *
947                                              (f_OSPLabor.Labor('PULLBOX',Tier_Flag)+
948                                               f_OSPMaterial.Mat('PULLBOX', 1));
949
950                                  if BedRock < 48
951                                    then duct_inv := duct_inv + ASmp.LnFtPullBox *
952   Round(Length/Asmp.ManHole_Spacing) *
953                                                     f_OSPLabor.Labor('LP70A',Tier_Flag);
954
955                                  if (InvType = 'AVG') and (Man_Pull_Cnt > 0)
956                                   {write manhole information to Inventory File }
957                                    then PartInventory.Write (Clli, OSPType[1], 'PULLBOX',
958   round(Length/Man_Pull_Cnt),
959                                                                Man_Pull_Cnt, Xpos, Ypos);
960                               end;  // End Place pullboxes
961                       end;    // End Copper cable placed
962             end;
963
964      {Determine if conduit investment or volume sensitive investment is to be
965       returned from the CalcCon function depending on the Inventory Type. }
966       if InvType = 'AVG' then
967         CalcCon := duct_Inv
968       else if InvType = 'VS' then
969         CalcCon := Vs_Conduit
970       else CalcCon := duct_Inv;
```

```
971
972      end; {end function CalCon}
973
974    begin   // True beginning of the Conduit function
975      {test to see if there is length for this placement}
976      Case ArcType of
977         0 : begin
978               OSPType := 'I';
979               Sharing_UndGnd := Asmp.Sharing_F_Cond;
980             end;
981         1 : begin
982               OSPType := 'F';
983               Sharing_UndGnd := Asmp.Sharing_F_Cond;
984             end;
985      2,3 : begin
986               if DM_ICM.t_Arcs.FieldByName('OSP Type').AsString = 'D'
987                 then begin
988                        OSPType := 'D';
989                        Sharing_UndGnd := Asmp.Sharing_D_Cond;
990                      end
991                 else begin
992                        OSPType := 'F';
993                        Sharing_Undgnd := Asmp.Sharing_F_Cond;
994                      end;
995            end;
996        4 : begin
997              OSPType := 'D';
998              Sharing_UndGnd := Asmp.Sharing_D_Cond;
999            end
1000     end;
1001
1002     if (Length < 1) or (Num_Of_Ducts < 1)
1003      then begin
1004             Result := 0;
1005             exit;    // then exit
1006           end;
1007
1008
1009      {Cost of material for conduit and manholes
```

```
1010        This sizes the number of ducts required in the conduit formation
1011        to care for the number of underground cables being placed.  The number
1012        of ducts (Num_Of_Ducts) is determined in the underground cable
1013        subroutine but the conduit formations come in standard sizes...2 ,4,
1014        6, 9, 12, 15, 18, 21, 24, etc.  This sizes the conduit formation for
1015        non-shared conduit.}
1016        if(Num_Of_Ducts/2 <= 3)
1017           then  size := round(2 * ceil(Num_Of_Ducts / 2.0))
1018           else  size := round(3 * ceil(Num_Of_Ducts / 3.0));
1019
1020        PartInventory.Enabled := False;  // RFPC 0012
1021        Norm_Inv := CalcCon;     // calculates the non-shared conduit investment
1022
1023        {This sizes the conduit formation for the share conduit, using the number
1024        of ducts for other companies and number of ducts required for the
1025        underground cables being placed.}
1026        if((Num_Of_Ducts + Asmp.ConduitUsers)/2 <= 3)
1027           then  size := round(2 * ceil((Num_Of_Ducts + Asmp.ConduitUsers) / 2.0))
1028           else  size := round(3 * ceil((Num_Of_Ducts + Asmp.ConduitUsers) / 3.0));
1029
1030        if InvType = 'AVG'         // RFPC 0012
1031          then PartInventory.Enabled := True   // RFPC 0012
1032          else PartInventory.Enabled := False; // RFPC 0012
1033
1034        Share_Inv := CalcCon;    // calculates the shared conduit investment
1035        {weigh the conduit, shared and non-shared, investment }
1036        PartInventory.Enabled := True;  // RFPC 0012
1037        Conduit := share_inv * Sharing_UndGnd *((Size - Asmp.ConduitUsers)/Size) +
1038                   Norm_inv * (1.0 - Sharing_UndGnd);
1039     end;  // END function Tf_grid.Conduit
1040
1041    function Poles( Clli : String; ArcType, Xpos, Ypos : integer;
1042                           Length : Double) : Double;
1043
1044     { This subroutine calculates the investment for poles required to support
1045       the aerial cables being placed.
1046
1047           Labor Activity Codes:  (used in Poles investment calculation)
1048           LP01A:      Place Pole/Push Brace (all sizes)
```

```
1049          LP03A:      Poles/Joint Use
1050          LP07A:      Place Anchor and Guy (all types/size)
1051          LP70A:      Solid Rock
1052          LP93C:      Cut and Remove Concrete (0 to 100 square feet)
1053          LP93D:      Place Concrete  (0 to 100 square feet)
1054    }
1055
1056    Var
1057      shpole_inv  : Double;      // shared pole investment
1058      nspole_inv  : Double;        // non-shared pole investment
1059      Pole_Cnt    : Double;        // total number of poles required for aerial cable
1060    begin      // start of Poles
1061       Case ArcType of
1062          0 : begin
1063                OSPType := 'I';
1064                Sharing_Aerial := Asmp.Sharing_F_Aerial;
1065                Foreign_Poles  := Asmp.Sharing_F_frgn_Poles
1066              end;
1067          1 : begin
1068                OSPType := 'F';
1069                Sharing_Aerial := Asmp.Sharing_F_Aerial;
1070                Foreign_Poles  := Asmp.Sharing_F_frgn_Poles
1071              end;
1072       2,3 : begin
1073                if DM_ICM.t_Arcs.FieldByName('OSP Type').AsString = 'D'
1074                  then begin
1075                         OSPType := 'D';
1076                         Sharing_Aerial := Asmp.Sharing_D_Aerial;
1077                         Foreign_Poles  := Asmp.Sharing_D_frgn_Poles
1078                       end
1079                  else begin
1080                         OSPType := 'F';
1081                         Sharing_Aerial := Asmp.Sharing_F_Aerial;
1082                         Foreign_Poles  := Asmp.Sharing_F_frgn_Poles
1083                       end;
1084              end;
1085          4 : begin
1086                OSPType := 'D';
1087                Sharing_Aerial := Asmp.Sharing_D_Aerial;
```

```
1088              Foreign_Poles  := Asmp.Sharing_D_frgn_Poles
1089            end
1090       end;
1091
1092      {test to see if there is aerial length for this placement}
1093       if Length < 1 then begin
1094         Result := 0;
1095         exit;    // if not, then exit
1096       end;
1097
1098      { POLES ARE VOLUME SENSITIVE}
1099      {calculate the number of poles required using length and distance between poles }
1100      pole_cnt := 1 + Round( Length / Asmp.Pole_Spacing ); // USF Coding Doc 3/25/98
1101      {reduce number of poles to reflect the percentage of foreign poles }
1102      pole_Cnt := round(Pole_Cnt * (1.0 - Foreign_Poles));
1103
1104      if Pole_Cnt < 1
1105        then begin
1106               Result := 0;
1107               exit;    // if not, then exit
1108             end;
1109
1110      {Bedrock within 60" of the surface  - non-shared pole investment (material
1111      adjusted for pole count, labor is not adjusted for pole count)
1112      (add cutting of bedrock)}
1113      if (Bedrock < 60 )
1114        then nspole_inv := pole_cnt * ( f_OSPMaterial.Mat('POLE',1)* (1+ f_OSPMaterial.Mat('MMPOLE',1)) +
1115                           f_OSPLabor.Labor('LP01A', Tier_Flag)+ Asmp.Percent_GuyWire *
1116                           (f_OSPLabor.Labor('LP07A', Tier_Flag) + f_OSPMaterial.Mat('ANCHOR',1)) +
1117                            Asmp.Percent_Concrete *
1118                           ( f_OSPLabor.Labor('LP93C', Tier_Flag)+ f_OSPLabor.Labor('LP93D', Tier_Flag))+
1119                           f_OSPLabor.Labor('LP70A', Tier_Flag))
1120
1121        {Bedrock >60" of the surface  - non-shared pole investment (material
1122        adjusted for pole count, labor is not adjusted for pole count)}
1123        else nspole_inv := pole_cnt * (f_OSPMaterial.Mat('POLE',1)* (1+ f_OSPMaterial.Mat('MMPOLE',1)) +
1124                           f_OSPLabor.Labor('LP01A', Tier_Flag)+ Asmp.Percent_GuyWire *
1125                           (f_OSPLabor.Labor('LP07A', Tier_Flag) + f_OSPMaterial.Mat('ANCHOR',1)) +
1126                            Asmp.Percent_Concrete *
```

```
1127                        (f_OSPLabor.Labor( 'LP93C',  Tier_Flag)+ f_OSPLabor.Labor( 'LP93D',  Tier_Flag)));
1128
1129      {Bedrock within 72" of the surface - shared pole investment (both material
1130      and labor are adjusted for pole count) (add cutting of bedrock)}
1131      if (Bedrock < 72 )
1132        then shpole_inv := pole_cnt * (f_OSPMaterial.Mat('POLESH',1)* (1+ f_OSPMaterial.Mat('MMPOLE',1))) +
1133                           pole_cnt * (f_OSPLabor.Labor('LP03A', Tier_Flag)+ Asmp.Percent_GuyWire *
1134                           (f_OSPLabor.Labor('LP07A', Tier_Flag)+ f_OSPMaterial.Mat('ANCHOR',1)) +
1135                            Asmp.Percent_Concrete *
1136                           (f_OSPLabor.Labor('LP93C', Tier_Flag) + f_OSPLabor.Labor('LP93D', Tier_Flag)) +
1137                           f_OSPLabor.Labor('LP70A', Tier_Flag ))
1138
1139        {Bedrock >72" of the surface  - shared pole investment (material adjusted
1140        for pole count, labor is not adjusted for pole count)}
1141        else shpole_inv := pole_cnt * (f_OSPMaterial.Mat('POLESH',1)* (1+ f_OSPMaterial.Mat('MMPOLE',1)) +
1142                           f_OSPLabor.Labor('LP03A', Tier_Flag)+ Asmp.Percent_GuyWire *
1143                           (f_OSPLabor.Labor('LP07A', Tier_Flag)+ f_OSPMaterial.Mat('ANCHOR',1)) +
1144                            Asmp.Percent_Concrete *
1145                           ( f_OSPLabor.Labor('LP93C', Tier_Flag)+ f_OSPLabor.Labor('LP93D', Tier_Flag)));
1146
1147      {Assume sharing with one other - weigh investments for non-shared and shared poles }
1148      Poles := shpole_inv * Sharing_Aerial / Asmp.PoleUsers  + nspole_inv * (1.0 - Sharing_Aerial);
1149      {Write poles information to Inventory File }
1150      if Length < Asmp.Pole_Spacing
1151        then PartInventory.Write (Clli, OSPType[1], 'POLES', Round(Length), round(pole_Cnt), Xpos, Ypos)
1152        else PartInventory.Write (Clli, OSPType[1], 'POLES', Round(Length/(Round(Length/Asmp.Pole_Spacing))),
1153                           round(pole_Cnt), Xpos, Ypos);
1154    end;       // END Poles function
1155
1156    function PairGainInvest(TDemand, RDemand, BDemand : Integer): Integer;
1157      Var
1158        K, L          : Longint;  // Total demand
1159        PGT           : Word;     // index to the material array
1160        PGC           : Word;     // pair gain's capacity
1161        PairGain_Cnt  : Word;     // pair gain count
1162        PG_Inv        : Double;   // sized invest
1163        rPG_Inv       : Double;   // remainder of investment
1164        VS_PGain      : Double;   // sized volume sensitive invest
1165        rVS_PGain     : Double;   // remainder of vs invest
```

```
1166       LabA, LabB    : String[15];
1167
1168     begin
1169        for PGT := 1 to 10 do //RFPC0030 run loop for 10 iterations
1170         begin
1171           K := round(TDemand * Asmp.Engineering_Feeder);
1172           L := f_OSPMaterial.MaxSize (DLCA[PGT]);
1173           if L = 0
1174            then begin
1175                  MessageDlg('Material ' + DLCA[PGT] +' not found.', mtError, [mbok], 0);
1176                  WeAreWatching.LogEvent(1,'Process Feeder','***ERROR*** Material ' + DLCA[PGT] +' not
1177   found.');
1178                end;
1179
1180           PairGain_Cnt := 0;
1181           PG_Inv       := 0.0;
1182           rPG_Inv      := 0.0;
1183           VS_PGain     := 0;
1184           rVS_PGain    := 0.0;
1185
1186           if K >= L then         // calc for DLC investment and DLC count
1187             begin
1188               LabA := DLCA[PGT] + InttoStr(L);
1189               LabB := DLCB[PGT] + InttoStr(L);
1190
1191               PGC := K div L;
1192               PairGain_Cnt := PGC;
1193               PG_Inv := ( f_OSPMaterial.Mat(DLCA[PGT], L) +
1194                           f_OSPLabor.Labor(LabA,  Tier_Flag) +
1195                           ( f_OSPMaterial.Mat(DLCB[PGT], L) +
1196                             f_OSPLabor.Labor(LabB,  Tier_Flag)) * L) * PGC;
1197
1198               if PerExtend > round(L / Asmp.Engineering_Feeder) * PGC
1199                 then begin
1200                        PG_Inv    := PG_Inv + f_OSPMaterial.Mat(DLCC[PGT], L) *
1201                                               round(L / Asmp.Engineering_Feeder);
1202                        PerExtend := PerExtend - round(L / Asmp.Engineering_Feeder) * PGC;
1203                      end
1204                 else begin
```

```
1205                    PG_Inv    := PG_Inv + f_OSPMaterial.Mat(DLCC[PGT], L) * PerExtend;
1206                    PerExtend := 0;
1207                  end;
1208
1209            VS_PGain :=  PG_Inv;
1210            K := K - (L * PGC);
1211            PartInventory.Write (Sai.Clli, 'F', DLCVar[PGT], L, PGC, SaiNode^.x, SaiNode^.y);
1212        end;
1213        if K > 0                // calc remainder
1214          then begin
1215                L := round(f_OSPMaterial.FindSize(DLCA[PGT], K));
1216                LabA := DLCA[PGT] + InttoStr(L);
1217                LabB := DLCB[PGT] + InttoStr(L);
1218
1219                Inc(PairGain_Cnt);
1220                rPG_Inv  := f_OSPMaterial.Mat(DLCA[PGT], L) +
1221                            f_OSPLabor.Labor(LabA,  Tier_Flag) +
1222                            ( f_OSPMaterial.Mat(DLCB[PGT], L) +
1223                              f_OSPLabor.Labor(LabB,  Tier_Flag))* k;
1224                rPG_Inv  := rPG_Inv + f_OSPMaterial.Mat(DLCC[PGT], L) * PerExtend;
1225
1226                rVS_PGain := ( ( f_OSPMaterial.Mat(DLCA[PGT], L) +
1227                                 f_OSPLabor.Labor(LabA,  Tier_Flag))/L +
1228                               f_OSPMaterial.Mat(DLCB[PGT],L) +
1229                               f_OSPLabor.Labor(LabB,  Tier_Flag)) * K;
1230                rVS_PGain := rVS_PGain + f_OSPMaterial.Mat(DLCC[PGT], L) * PerExtend;
1231                PartInventory.Write (Sai.Clli, 'F', DLCVar[PGT], L, 1, SaiNode^.x, SaiNode^.y);
1232             end;
1233
1234        PG_Inv   := PG_Inv + rPG_Inv;
1235        VS_PGain := VS_PGain + rVS_PGain;
1236        OSP_Table.Write(Sai.CLLI, '223200', 'TF'+DLCVar[PGT], 'F', PG_Inv, VS_PGain, PG_Inv - VS_PGain);
1237        OSP_Table.Write(Sai.CLLI, '223200', 'RF'+DLCVar[PGT], 'F', PG_Inv    * RDemand / TDemand,
1238                                                                   VS_PGain * RDemand / TDemand,
1239                                                                  (PG_Inv-VS_PGain) * RDemand / TDemand);
1240        OSP_Table.Write(Sai.CLLI, '223200', 'BF'+DLCVar[PGT], 'F', PG_Inv  * BDemand / TDemand,
1241                                                                   VS_PGain * BDemand / TDemand,
1242                                                                  (PG_Inv-VS_PGain) * BDemand / TDemand);
1243      end;
```

```
1244
1245      PairGainInvest := PairGain_Cnt;
1246    end;
1247
1248  Function XConnInvest(Clli : String; X, Y, TD : Integer) : Double;
1249    var
1250      Tmp_Demand  : Longint;
1251      Box_Cnt     : Longint;
1252      S           : Longint;
1253      SD          : Longint;
1254      Splices     : Double;
1255      Inv         : Double;
1256      rInv        : Double;
1257
1258    begin
1259      Inv := 0.0;
1260      rInv:= 0.0;
1261      SD := TD * 2;
1262
1263      {Adjust demand based on the Xconn factor }
1264      Tmp_Demand := round(TD * Asmp.XConn_Factor);
1265      S := f_OSPMaterial.MaxSize('XCONN');
1266      if S > 0
1267        then begin
1268               { How many cross connect boxes do we need }
1269               Box_cnt:= Tmp_demand Div S;
1270               if Box_cnt > 0
1271                 then begin
1272                        Inv := (f_OSPMaterial.Mat('XCONN', s)+
1273                                f_OSPLabor.Labor('LP28A', Tier_Flag) +
1274                                f_OSPLabor.Labor('LP28D', Tier_Flag)) * Box_cnt;
1275
1276                        // Splicing   2 X demand
1277                        Splices := S * 2.0/Asmp.XConn_Factor * Box_cnt;
1278                        Case SD of
1279                           1..50 : Inv := Inv + f_OSPLabor.Labor('LS02A', Tier_Flag) * Splices;
1280                          51..300 : Inv := Inv + f_OSPLabor.Labor('LS02B', Tier_Flag) * Splices;
1281                             else   Inv := Inv + f_OSPLabor.Labor('LS02C', Tier_Flag) * Splices;
1282                        end;
```

```
1283                        // Jumpers  1 X demand
1284                        Splices := S /Asmp.XConn_Factor * Box_cnt;
1285                        Inv := Inv + f_OSPLabor.Labor('LS20A', Tier_Flag) * Splices;
1286
1287                        Tmp_Demand := Tmp_Demand - (S * Box_cnt);
1288                        PartInventory.Write (Clli, 'F', 'XCONN', S, Box_cnt, X, Y);
1289                      end;
1290              end
1291          else begin
1292                MessageDlg('Cross connect boxes not available', mterror, [mbOk], 0);
1293                exit;
1294              end;
1295        if Tmp_Demand > 0
1296          then begin
1297                s   := round(f_OSPMaterial.FindSize('XCONN', Tmp_Demand));
1298                rInv := f_OSPMaterial.Mat('XCONN', s)+
1299                        f_OSPLabor.Labor('LP28A', Tier_Flag) +
1300                        f_OSPLabor.Labor('LP28D', Tier_Flag);
1301
1302                // Splicing   2 X demand
1303                Splices := Tmp_Demand * 2.0/Asmp.XConn_Factor;
1304                Case SD of
1305                   1..50 : rInv := rInv + f_OSPLabor.Labor('LS02A', Tier_Flag) * Splices;
1306                  51..300 : rInv := rInv + f_OSPLabor.Labor('LS02B', Tier_Flag) * Splices;
1307                     else   rInv := rInv + f_OSPLabor.Labor('LS02C', Tier_Flag) * Splices;
1308                end;
1309                // Jumpers  1 X demand
1310                Splices := Tmp_Demand /Asmp.XConn_Factor;
1311                rInv := rInv + f_OSPLabor.Labor('LS20A', Tier_Flag) * Splices;
1312                PartInventory.Write (Clli, 'F', 'XCONN', S, 1, X, Y);
1313              end;
1314
1315      Result := Inv + rInv;
1316    end;
1317
1318  Function Place_CU_Aerial(Clli : String; ArcType, Xpos, Ypos : integer;
1319                                      Length, Demand : Longint; UFill : Double) : Double;
1320
1321    { This subroutine calculates the investments for placing aerial copper cables
```

```
1322
1323        Labor activity codes: (used in Place_CU_Aerial)
1324              LP18A:      Place Strand (all sizes)
1325              LP22A:          Place aerial cable (all types/sizes)
1326              LS02A:          Straight Splice (1-50 pairs)
1327              LS02B:      Straight Splice (51-300 pairs)
1328              LS02C:      Straight Splice (>300 pairs)
1329    }
1330
1331     Var
1332       Inv, C_Inv, rC_Inv, pf   : Double;
1333       CS, PairCnt, tmp_Demand : LongInt;
1334       Total_Splices           : LongInt;
1335       SCable                  : String[15];
1336
1337    begin                      // set initial values
1338       Inv      := 0.0;        // strand and placing investment
1339       C_Inv    := 0.0;        // total / maximum size cable investment
1340       rC_Inv   := 0.0;        // less than maximum size cable investment
1341       pf       := 0.0;        // pair feet
1342       PairCnt  := 0;          // pair count
1343       Cable_Cnt:= 0;          // number of cables
1344
1345       Case ArcType of
1346          0 : OSPType := 'I';
1347          1 : OSPType := 'F';
1348        2,3 : OSPType := DM_ICM.t_Arcs.FieldByName('OSP Type').AsString;
1349          4 : OSPType := 'D';
1350       end;
1351
1352       {This section sets the gauge required for the distribution cables to 24 gauge
1353       when the 18 KFT option is selected by the user.}
1354       if (ArcType > 1) and (CUGA = 24)
1355        then SCable := 'CuAer24'
1356        else SCable := 'CuAer26';
1357
1358       {test to see if there is a demand and length for aerial copper cable placement}
1359       if (Demand < 1) or (Length < 1) then begin
1360         Result := 0;
```

```
1361        exit;    // if not, then exit
1362        WeAreWatching.LogEvent(mFUNC,'Place_Cu_Aerial','  ('+Clli+','+IntToStr(Xpos)+','+intTostr(Ypos)+
1363                          ','+IntToStr(Length)+','+IntToStr(demand)+
1364                          ','+FloatToStr(UFill)+'=0 (Demand<1, Length<1)');
1365      end;
1366
1367      {This section determines the temporary demands for sizing the copper cables}
1368      case ArcType  of
1369          1 : tmp_demand := Round(Demand * Asmp.Engineering_Feeder);          // feeder
1370        2,3 : if DM_ICM.t_Arcs.FieldByName('OSP Type').AsString = 'D'
1371                 then tmp_demand := Round(Demand * Asmp.Engineering_Dist)     // distribution
1372                 else tmp_demand := Round(Demand * Asmp.Engineering_Feeder);  // feeder
1373          4 : tmp_demand := Round(Demand * Asmp.Engineering_Dist);           // distribution
1374       else tmp_demand := Demand;                                            // all other
1375      end;
1376
1377      {This section determines if the demand requires the largest cable size
1378      and if so how many.}
1379      CS := f_OSPMaterial.MaxSize(SCable); // get largest aerial copper cable in the material file
1380      if CS > 0     // check for positive value as precaution
1381        then begin
1382               {How many maximum size cables do we need to meet the temporary demand}
1383               cable_cnt:= Trunc(tmp_demand/(CS*Asmp.ADMNFILL));
1384               if cable_cnt > 0       // at least one maximum size cable is required
1385                 then begin   { Strand and labor added for very large cables --- September 4 }
1386                        PairCnt := PairCnt + CS * cable_cnt;
1387                        C_Inv      := f_OSPMaterial.Mat(SCable, CS) * Length * cable_cnt;
1388                        pf         := pf + cable_cnt * Length * CS;
1389                        tmp_demand := tmp_demand - round(cable_cnt * CS * Asmp.ADMNFILL);
1390                        {Write information on the aerial copper to the Inventory
1391                        File.  When a full size cable is placed the segment
1392                        number is written as a negative to the inventory file.
1393                        A positive segment number is used for the smaller cables.}
1394                        PartInventory.Write (Clli, OSPType[1], SCable, CS, Length*cable_Cnt, Xpos, Ypos);
1395                      end;
1396           end
1397        else begin  // display error if no aerial copper cable in the material file
1398               MessageDlg('Copper Aerial not available', mterror, [mbOk], 0);
1399          //     WeAreWatching.LogEvent(mERROR,'Place_CU_Aerial','***'+Clli+' Copper Aerial not available');
```

```
1400                Result := 0.0;
1401                exit;
1402              end;
1403
1404          if tmp_Demand > 0
1405           then begin
1406                  {This section develops the investments for aerial cables when
1407                  maximum size cables are not required or the remaining temporary
1408                  demand after the demand cared for by the maximum cables placed
1409                  in the above section has been subtracted.}
1410                  CS     := f_OSPMaterial.FindSize(SCable, round(tmp_Demand/Asmp.ADMNFILL));
1411                  Inc(Cable_cnt); // increment cable count to include all maximum size cables
1412                  PairCnt:= PairCnt + CS;
1413                  rC_Inv := f_OSPMaterial.Mat(SCable, CS) * Length;
1414                  pf     := pf + CS * Length;
1415                  {Write information on the aerial copper to the Inventory File.
1416                  Note the positive segment.}
1417                  PartInventory.Write (Cl1i, OSPType[1], SCable, CS, Length, Xpos, Ypos);
1418               end;
1419
1420     C_Inv := C_Inv + rC_Inv;   // total investment (max + less than max size cables)
1421
1422     { This section calculates the number of splices (in pairs) required.}
1423     { The feeder splicing is based on the total number of pairs to be spliced.}
1424     Splice_Pts := Round(length/Asmp.Aer_Cu_Span);
1425     if Splice_Pts = 0 then Splice_Pts := 1;
1426     Total_Splices := PairCnt * Splice_Pts;
1427
1428     {The labor rate for splicing varies by the number of pairs being spliced.}
1429     Case PairCnt of
1430       1..50 : C_Inv := C_Inv + f_OSPLabor.Labor('LS02A', Tier_Flag) * Total_Splices;
1431     51..300 : C_Inv := C_Inv + f_OSPLabor.Labor('LS02B', Tier_Flag) * Total_Splices;
1432        else   C_Inv := C_Inv + f_OSPLabor.Labor('LS02C', Tier_Flag) * Total_Splices;
1433     end;
1434   {------------------------}
1435
1436     {Calculate the investment for strand and the labor for placing the strand
1437     and cable.  This investment is kept separate from the cable investment
1438     until after the user fill option. A strand is placed for each cable because
```

```
1439      multiple cables are only placed when maximum size cables are involved and
1440      only 1 maximum size cable per strand.}
1441      {Strand added to the cost calculation-- September 4 }
1442      Inv     := Inv + (f_OSPMaterial.Mat('STRAND',1)+
1443                        f_OSPLabor.Labor('LP18A', Tier_Flag)+
1444                        f_OSPLabor.Labor('LP22A', Tier_Flag))* Length * Cable_Cnt; {2/25/98}
1445      {Write information on the strand to the Inventory File. }
1446      PartInventory.Write (Clli, OSPType[1], 'STRAND', 1, Length * Cable_Cnt, Xpos, Ypos);
1447
1448       {This section is activated when the user clicks on the "User Fill Option"
1449       and specifies the fills for the feeder and distribution that they want used. }
1450       if Asmp.UserFill and (pf > 0.0) and (Ufill > 0.00001)
1451        then begin
1452               {cable investment is adjusted base on the ratio of the actual and user fill factors}
1453               C_Inv := C_Inv * ((Demand * Length / pf) / UFill);
1454               {Capacity pair feet is calculated base on the user fill factor}
1455               pf    := (Demand * Length) / UFill;
1456             end;
1457
1458      Pair_Ft_Cap:= Pair_Ft_Cap + pf;  // pair feet of cable placed (pair_ft_cap)
1459      Pair_Ft_Utl:= Pair_Ft_Utl + Demand * Length; // pair feet of cable utilized (pair_ft_utl)
1460
1461      {Combine the cable and splicing investment (C_Inv) with the strand and
1462      placing investment (Inv) to get total investment for aerial copper cable. }
1463      Result := Inv + C_Inv;
1464      WeAreWatching.LogEvent(mFUNC,'Place_CU_Aerial','  ('+Clli+','+IntToStr(Xpos)+','+intTostr(Ypos)+
1465                      ','+IntToStr(Length)+','+IntToStr(demand)+
1466                      ','+FloatToStr(UFill)+'='+FloatToStr(Result));
1467  end;  // END Place_CU_Aerial function
1468
1469  Function Place_CU_Buried(Clli : String; ArcType, Xpos, Ypos : integer;
1470                                  Length, Demand : Longint; UFill : Double) : Double;
1471
1472   { This subroutine calculates the investment for placing buried copper cables
1473
1474        Labor activity codes: (used in Place_CU_Buried)
1475              LP49B:          Place additional cable/duct
1476              LP73A:          Splice Pit
1477              LS02A:          Straight Splice (1-50 pairs)
```

```
1478              LS02B:      Straight Splice (51-300 pairs)
1479              LS02C:      Straight Splice (>300 pairs)
1480    }
1481
1482     Var
1483       Inv, C_Inv, rC_Inv, pf : Double;
1484       CS, PairCnt, tmp_Demand : LongInt;
1485       Total_Splices           : LongInt;      // number of splices (distrib or feeder)
1486       SCable                  : String[15];   // type and gauge of cable
1487
1488     begin                // set initial values
1489       Inv      := 0.0;        // strand and placing investment
1490       C_Inv    := 0.0;        // cable investment for max size cable
1491       rC_Inv   := 0.0;        // cable investment for less than max size cable
1492       pf       := 0.0;        // pair feet
1493       PairCnt  := 0;          // number of pair
1494       Cable_Cnt:= 0;          // number of cables
1495
1496       Case ArcType of
1497          0 : OSPType := 'I';
1498          1 : OSPType := 'F';
1499        2,3 : OSPType := DM_ICM.t_Arcs.FieldByName('OSP Type').AsString;
1500          4 : OSPType := 'D';
1501       end;
1502
1503       {This section sets the gauge required for the distribution cables to 24 when
1504       the 18 KFT option is selected by the user.  The 24 gauge is placed in the
1505       four grids located in the 4 corners of the 6x6 super cells.  That means that
1506       24 gauge is placed in 16 of the 26 grids in the 6x6 super cell. }
1507       if (ArcType > 1) and (CUGA = 24)
1508        then SCable := 'CuBur24'
1509        else SCable := 'CuBur26';
1510
1511       {Test to see if there is a demand and length for buried copper placement}
1512       if (Demand < 1) or (Length < 1) then begin
1513         Result := 0;
1514         exit;    // if not, then exit
1515         WeAreWatching.LogEvent(mFUNC,'Place_Cu_Buried','  ('+Clli+','+IntToStr(Xpos)+','+intTostr(Ypos)+
1516                              ','+IntToStr(Length)+','+IntToStr(demand)+
```

```
1517                        ','+FloatToStr(UFill)+'=0 (Demand<1, Length<1)');
1518      end;
1519
1520      {This section determines the temporary demands for sizing the copper cables}
1521      case ArcType  of
1522          1 : tmp_demand := Round(Demand * Asmp.Engineering_Feeder);          // feeder
1523        2,3 : if DM_ICM.t_Arcs.FieldByName('OSP Type').AsString = 'D'
1524                  then tmp_demand := Round(Demand * Asmp.Engineering_Dist)     // distribution
1525                  else tmp_demand := Round(Demand * Asmp.Engineering_Feeder);  // feeder
1526          4 : tmp_demand := Round(Demand * Asmp.Engineering_Dist);            // distribution
1527       else tmp_demand := Demand;                                             // all other
1528      end;
1529
1530      {This section determines if the demand requires the largest cable size and
1531      if so, how many. }
1532      CS := f_OSPMaterial.MaxSize(SCable); // largest buried copper cable in material file
1533      if CS > 0           // check for max cable found as precaution
1534        then begin
1535               {determine how many max size cables required to meet the
1536               temporary demand }
1537               cable_cnt:= Trunc(tmp_demand/(CS*Asmp.ADMNFILL));
1538               if cable_cnt > 0  // are there any max size cables required?
1539                 then begin
1540                        {One max size cable is required so calculate using the
1541                        max size cable count, the max size cable investment, the
1542                        pair count, the pair feet, and the temporary demand for
1543                        cables that are less than the max size.  Write the
1544                        information on the buried copper cable to the inventory
1545                        file (when a full size cable is placed the segment number
1546                        is written as a negative to the inventory file.  A
1547                        positive segment number is used for the smaller cables.) }
1548                        C_Inv  := f_OSPMaterial.Mat(SCable, CS) *
1549                                  Length * cable_cnt;
1550                        PairCnt := PairCnt + CS * Cable_Cnt;
1551                        pf     := pf + cable_cnt * Length * CS;
1552                        tmp_demand := tmp_demand - round(cable_cnt * CS * Asmp.ADMNFILL);
1553                        PartInventory.Write (Clli, OSPType[1], SCable, CS, Cable_Cnt*Length, Xpos, Ypos);
1554                      end;
1555           end
```

```
1556        else begin
1557               {no max size cables were found in the material file, so display
1558               an error message }
1559               MessageDlg('Copper Buried not available', mterror, [mbOk], 0);
1560               Result:= 0.0;
1561               WeAreWatching.LogEvent(mERROR,'Place_CU_Buried','***'+Clli+' Copper Buried not available');
1562               exit;
1563             end;
1564
1565       {This section develops the investments for buried cables when max size cables
1566       are not required or the temporary demand after the demand cared for by the
1567       max cables placed, in the above section, has been subtracted. }
1568       if tmp_Demand > 0
1569         then begin
1570                CS := f_OSPMaterial.FindSize(SCable, round(tmp_Demand/Asmp.ADMNFILL));
1571                Inc (Cable_Cnt);
1572                PairCnt := PairCnt + CS;
1573                rC_Inv := f_OSPMaterial.Mat(SCable, CS) * Length;
1574                pf     := pf + CS * Length;
1575                PartInventory.Write (Clli, OSPType[1], SCable, CS, Length, Xpos, Ypos);
1576              end;
1577
1578      {calculate total (material) investment using investment for max size
1579      cables and the investment for less than max size cables }
1580      C_Inv := C_Inv + rC_Inv;
1581
1582     { 2/25/98 }
1583     Splice_Pts := Round(length/Asmp.Bur_Cu_Span);
1584     if Splice_Pts = 0 then Splice_Pts := 1;
1585
1586      if DM_ICM.t_Arcs.FieldByName('OSP Type').AsString = 'F'
1587        {Add the investment for splice pits since the cable being processed
1588        is not a distribution cable. }
1589        then Inv := Inv + f_OSPLabor.Labor('LP73A',  Tier_Flag) * Splice_Pts; // 7/23/98
1590
1591     Total_Splices := PairCnt * Splice_Pts;
1592
1593     {The labor rate for splicing varies by the number of pairs being spliced.}
1594     Case PairCnt of
```

```
1595        1..50 : C_Inv := C_Inv + f_OSPLabor.Labor('LS02A', Tier_Flag) * Total_Splices;
1596       51..300 : C_Inv := C_Inv + f_OSPLabor.Labor('LS02B', Tier_Flag) * Total_Splices;
1597          else   C_Inv := C_Inv + f_OSPLabor.Labor('LS02C', Tier_Flag) * Total_Splices;
1598       end;
1599     {------------------------}
1600
1601        {if user wants to use a specific fill factor }
1602        if Asmp.UserFill and (pf > 0.0) and (Ufill > 0.00001)
1603          then begin
1604                 {cable investment is adjusted base on the ratio of the actual and user fill factors}
1605                 C_Inv := C_Inv * ((Demand * Length / pf) / UFill);
1606                 {Capacity pair feet is calculated base on the user fill factor}
1607                 pf    := (Demand * Length) / UFill;
1608               end;
1609
1610        if Cable_Cnt > 1
1611          {more than one cable is being placed so add additional labor to
1612          investment dependant on the cable count }
1613          then C_Inv := C_Inv + f_OSPLabor.Labor('LP49B', Tier_Flag) * (Cable_Cnt-1) * Length;
1614
1615        Pair_Ft_Cap:= Pair_Ft_Cap + pf; // pair feet of cable placed (pair_ft_cap)
1616        Pair_Ft_Utl:= Pair_Ft_Utl + Demand * Length; // pair feet of cable utilized (pair_ft_utl)
1617
1618        {calculate the returned buried copper cable investment value using the
1619        cable and splicing investment (C_Inv) and the strand and placing investment
1620        (Inv) }
1621        Result := Inv + C_Inv;
1622        WeAreWatching.LogEvent(mFUNC,'Place_CU_Buried','  ('+Clli+','+IntToStr(Xpos)+','+intTostr(Ypos)+
1623                     ','+IntToStr(Length)+','+IntToStr(demand)+
1624                     ','+FloatToStr(UFill)+'='+FloatToStr(Result));
1625
1626     end;       // END Place_CU_Buried function
1627
1628   Function Place_CU_UndGnd(Clli : String; ArcType, Xpos, Ypos : integer;
1629                                  Length, Demand : Longint; UFill : Double) : Double;
1630
1631     { This subroutine calculates the investments for placing underground copper cables
1632
1633          Labor activity codes: (used in Place_CU_UndGnd)
```

```
1634              LP43A:      Place copper cable in conduit
1635              LS02A:          Straight Splice (1-50 pairs)
1636              LS02B:      Straight Splice (51-300 pairs)
1637              LS02C:      Straight Splice (>300 pairs)
1638              LS72A:      Manhole setup
1639    )
1640
1641     Var
1642       Inv, C_Inv, rC_Inv, pf  : Double;
1643       CS, PairCnt, tmp_Demand : LongInt;
1644       Total_Splices           : Longint;
1645       SCable                  : String[15];
1646
1647     begin                     // set initial values
1648       Inv      := 0.0;        // placing investment
1649       C_Inv    := 0.0;        // cable and splicing investment
1650       rC_Inv   := 0.0;        // less than max size cable investment
1651       pf       := 0.0;        // pair feet
1652       PairCnt  := 0;          // number of pair
1653       Cable_Cnt:= 0;          // number of cables
1654
1655       Case ArcType of
1656          0 : OSPType := 'I';
1657          1 : OSPType := 'F';
1658        2,3 : OSPType := DM_ICM.t_Arcs.FieldByName('OSP Type').AsString;
1659          4 : OSPType := 'D';
1660       end;
1661
1662       {This section sets the gauge required for the distribution cables to 24 when
1663       the 18 KFT option is selected by the user.  The 24 gauge is placed in the
1664       four grids located in the 4 corners of the 6x6 super cells.  That means that
1665       24 gauge is placed in 16 of the 26 grids in the 6x6 super cell. }
1666       if (ArcType > 1) and (CUGA = 24)
1667        then SCable := 'CuUnd24'
1668        else SCable := 'CuUnd26';
1669
1670       {test to see if there is a demand and length for underground copper
1671       cable placement}
1672       if (Demand < 1) or (Length < 1) then begin
```

```
1673        Result := 0;
1674        exit;    // if not, then exit
1675        WeAreWatching.LogEvent(mFUNC,'Place_Cu_UndGnd',' ('+Clli+','+IntToStr(Xpos)+','+intTostr(Ypos)+
1676                        ','+IntToStr(Length)+','+IntToStr(demand)+
1677                        ','+FloatToStr(UFill)+'=0 (Demand<1, Length<1)');
1678      end;
1679
1680      {This section determines the temporary demands for sizing the copper cables}
1681      case ArcType  of
1682          1 : tmp_demand := Round(Demand * Asmp.Engineering_Feeder);          // feeder
1683        2,3 : if DM_ICM.t_Arcs.FieldByName('OSP Type').AsString = 'D'
1684                  then tmp_demand := Round(Demand * Asmp.Engineering_Dist)     // distribution
1685                  else tmp_demand := Round(Demand * Asmp.Engineering_Feeder);  // feeder
1686          4 : tmp_demand := Round(Demand * Asmp.Engineering_Dist);            // distribution
1687       else tmp_demand := Demand;                                             // all other
1688      end;
1689
1690      {This section determines if the demand requires the largest cable size and
1691      if so, how many. }
1692      CS := f_OSPMaterial.MaxSize(SCable);  // largest underground copper cable in material file
1693      if CS > 0           // check for max cable found as precaution
1694        then begin
1695               { determine how many max size cables required to meet the
1696               temporary demand)
1697               cable_cnt:= Trunc(tmp_demand/(CS*Asmp.ADMNFILL));
1698               if cable_cnt > 0  // are there any max size cables required?
1699                 then begin
1700                        {One max size cable is required so calculate using the
1701                        max size cable count, the max size cable investment, the
1702                        pair count, the pair feet, and the temporary demand for
1703                        cables that are less than the max size.  Write the
1704                        information on the underground copper cable to the
1705                        inventory file (when a full size cable is placed the
1706                        segment number is written as a negative to the inventory
1707                        file.  A positive segment number is used for the smaller
1708                        cables.) }
1709                        C_Inv      := f_OSPMaterial.Mat(SCable, CS) *
1710                                      Length * cable_cnt;
1711                        PairCnt:= PairCnt + CS * cable_cnt;
```

```
1712                        pf          := pf + cable_cnt * Length * CS;
1713                        tmp_demand := tmp_demand - round(cable_cnt * CS * Asmp.ADMNFILL);
1714                        PartInventory.Write (Clli, OSPType[1], SCable, CS, Cable_Cnt*Length, Xpos, Ypos);
1715                      end;
1716              end
1717         else begin
1718                {no max size cables were found in the material file, so display
1719                an error message }
1720                MessageDlg('Copper undergound not available', mterror, [mbOk], 0);
1721                Result := 0.0;
1722                WeAreWatching.LogEvent(mERROR,'Place_CU_UndGnd','***'+Clli+' Copper Underground not
1723   available');
1724                exit;
1725              end;
1726
1727        {This section develops the investments for underground cables when max
1728        size cables are not required or the temporary demand after the demand
1729        cared for by the max cables placed, in the above section, has been
1730        subtracted. }
1731        if tmp_Demand > 0
1732          then begin
1733                 CS     := f_OSPMaterial.FindSize(SCable, round(tmp_Demand/Asmp.ADMNFILL));
1734                 Inc (Cable_Cnt);
1735                 PairCnt:= PairCnt + CS;
1736                 rC_Inv := f_OSPMaterial.Mat(SCable, CS) * Length;
1737                 pf     := pf + CS * Length;
1738                 PartInventory.Write (Clli, OSPType[1], SCable, CS, Length, Xpos, Ypos);
1739               end;
1740
1741        {calculate total (material) investment using investment for max size
1742        cables and the investment for less than max size cables }
1743        C_Inv := C_Inv + rC_Inv;
1744
1745        {calculate the investment for placing underground cable using the labor
1746        for placing copper cable in a conduit }
1747        Inv := Inv + f_OSPLabor.Labor('LP43A',  Tier_Flag) * Length * Cable_Cnt;
1748        {when underground cable is being placed, the investment for
1749         manhole site setup is added for each manhole }
1750
```

```
1751        Splice_Pts := Round(length/Asmp.Manhole_Spacing);
1752        if Splice_Pts = 0 then Splice_Pts := 1;
1753        Inv := Inv + f_OSPLabor.Labor('LS72A',  Tier_Flag) * Splice_Pts;
1754
1755       {calculate the number of splices required }
1756       Total_Splices := PairCnt * Splice_Pts;
1757
1758      {The labor rate for splicing varies by the number of pairs being spliced.}
1759      Case PairCnt of
1760        1..50 : C_Inv := C_Inv + f_OSPLabor.Labor('LS02A', Tier_Flag) * Total_Splices;
1761      51..300 : C_Inv := C_Inv + f_OSPLabor.Labor('LS02B', Tier_Flag) * Total_Splices;
1762         else   C_Inv := C_Inv + f_OSPLabor.Labor('LS02C', Tier_Flag) * Total_Splices;
1763      end;
1764    {------------------------}
1765
1766      { if user wants to use a specific fill factor }
1767       if Asmp.UserFill and (pf > 0.0) and (Ufill > 0.00001)
1768          then begin
1769                 { cable investment is adjusted base on the ratio of the actual and user fill factors}
1770                 C_Inv := C_Inv * ((Demand * Length / pf) / UFill);
1771                 { Capacity pair feet is calculated base on the user fill factor}
1772                 pf    := (Demand * Length) / UFill;
1773               end;
1774
1775        Pair_Ft_Cap:= Pair_Ft_Cap + pf;       // pair feet of cable placed (pair_ft_cap)
1776        Pair_Ft_Utl:= Pair_Ft_Utl + Demand * Length; // pair feet of cable utilized (pair_ft_utl)
1777
1778        {calculate the returned underground copper cable investment value using
1779        the cable and splicing investment (C_Inv) and the placing investment (Inv) }
1780        Result := Inv + C_Inv;
1781        WeAreWatching.LogEvent(mFUNC,'Place_CU_UndGnd','  ('+Clli+','+IntToStr(Xpos)+','+intTostr(Ypos)+
1782                     ','+IntToStr(Length)+','+IntToStr(demand)+
1783                     ','+FloatToStr(UFill)+'='+FloatToStr(Result));
1784     end;      // END Place_CU_UndGnd function
1785
1786     Function Place_FI_Aerial(Clli : String; ArcType, Xpos, Ypos : integer;
1787                                    Length, Strands, Demand : Longint; UFill : Double) : Double;
1788
1789    { This subroutine calculates the investments for placing aerial fiber cables.
```

```
1790
1791        Labor activity codes:  (used in Place_FI_Aerial)
1792              LP18A:          Place strand (all sizes)
1793              LP22A:      Place aerial cable (all types/size)
1794              LS50A:      Splice Fiber Optic cable - 48 fibers or less
1795              LS50B:          Splice Fiber Optic cable - greater than 48 fibers
1796    }
1797
1798     Var
1799       Inv, C_Inv, rC_Inv, pf : Double;
1800       CS, tmp_Strands  : LongInt;
1801       Total_Fi_Strands : Longint;
1802
1803     begin                       // set initial values
1804       Inv      := 0.0;          // the strand and placing investment
1805       C_Inv    := 0.0;          // investment for max size cables
1806       rC_Inv   := 0.0;          // investment for less than max size cables
1807       Cable_Cnt:= 0;            // number of cables
1808       Total_Fi_Strands := 0;    // number of fiber strands
1809
1810       Case ArcType of
1811          0 : OSPType := 'I';
1812          1 : OSPType := 'F';
1813        2,3 : begin
1814                if DM_ICM.t_Arcs.FieldByName('OSP Type').AsString = 'D'
1815                  then OSPType := 'D'
1816                  else OSPType := 'F';
1817              end;
1818          4 : OSPType := 'D';
1819       end;
1820
1821       {test to see if there is a demand and length for aerial fiber placement}
1822       if (Demand < 1) or (Length < 1) then begin
1823         Result := 0;
1824         WeAreWatching.LogEvent(mFUNC,'Place_Fi_Aerial','  ('+Clli+','+IntToStr(Xpos)+','+intTostr(Ypos)+
1825                           ','+IntToStr(Length)+','+IntToStr(strands)+','+IntToStr(demand)+
1826                           ','+FloatToStr(UFill)+'=0 (Demand<1, Length<1)');
1827         exit;    // if not, then exit
1828       end;
```

```
1829
1830      tmp_Strands := Strands;          // set temporary strands
1831      CS := f_OSPMaterial.MaxSize('FiAer');  // largest aerial fiber cable size in material table
1832      if CS > 0           // check for max cable found as precaution
1833        then begin
1834               {calculate the number of cables required to meet the temporary
1835               demand}
1836               cable_cnt:= Trunc(tmp_Strands/CS);
1837               if cable_cnt > 0  // are there any max size cables required?
1838                 then begin
1839                        {At least one max size cable is required so calculate
1840                        investment using the max size cable count.  Write the
1841                        information on the aerial fiber cable to the inventory
1842                        file (when a full size cable is placed the segment
1843                        number is written as a negative to the inventory file.
1844                        A positive segment number is used for the smaller cables.) }
1845                        C_Inv      := f_OSPMaterial.Mat('FiAer', CS) * Length * cable_cnt;
1846                        tmp_Strands := tmp_Strands - round(cable_cnt * CS);
1847                        Total_Fi_Strands := Total_Fi_Strands + cable_cnt * CS;
1848                        PartInventory.Write (Clli, OSPType[1], 'AERIALFI', CS, Cable_Cnt*Length, Xpos, Ypos);
1849                      end;
1850             end
1851        else begin
1852               {no max size cables were found in the material file, so display
1853               an error message }
1854               MessageDlg('Fiber Aerial not available', mterror, [mbOk], 0);
1855               Result := 0.0;
1856               WeAreWatching.LogEvent(mERROR,'Place_Fi_Aerial','***'+Clli+' Fiber Aerial not available');
1857               exit;
1858             end;
1859
1860        if tmp_Strands > 0
1861         then begin
1862                {Find cable size for strands or remaining strands }
1863                CS     := f_OSPMaterial.FindSize('FiAer', tmp_Strands);
1864                Inc (Cable_Cnt);
1865                rC_Inv  := f_OSPMaterial.Mat('FiAer', CS) * Length;
1866                Total_Fi_Strands := Total_Fi_Strands + CS;
1867                PartInventory.Write (Clli, OSPType[1], 'AERIALFI', CS, Length, Xpos, Ypos);
```

```
1868              end;
1869
1870     {calculate total (material) investment using investment for max size cables
1871     and the investment for less than max size cables }
1872     C_Inv := C_Inv + rC_Inv;
1873
1874    // USF Coding Doc 3/25/98
1875    Splice_Pts := Round(length/Asmp.Aer_Fi_Span);
1876    if Splice_Pts = 0 then Splice_Pts := 1;
1877
1878    if Arctype < 2
1879      then begin
1880             if Total_Fi_Strands < 49
1881               {calculate the splicing investment using labor dependant on the
1882               total number of fiber strands }
1883               then C_Inv := C_Inv + f_OSPLabor.Labor('LS50A', Tier_Flag) * Total_Fi_Strands * Splice_Pts
1884               else C_Inv := C_Inv + f_OSPLabor.Labor('LS50B', Tier_Flag) * Total_Fi_Strands * Splice_Pts;
1885           end;
1886    {--------------------}
1887
1888     {calculate the investment for strand material and labor for placing
1889     the strand and aerial fiber cable }
1890     {Strand added to the cost calculation-- September 4 }
1891     Inv    := Inv + (f_OSPMaterial.Mat('STRAND',1)+
1892                      f_OSPLabor.Labor('LP18A', Tier_Flag)+
1893                      f_OSPLabor.Labor('LP22A', Tier_Flag))* Length;
1894     {Write the strand information on the aerial fiber cable to the inventory file }
1895     PartInventory.Write (Clli, OSPType[1], 'STRAND', 1, Length, Xpos, Ypos);
1896
1897     pf := (Demand * Length) /(Strands/Total_Fi_Strands);
1898      {if user wants to use a specific fill factor }
1899      if Asmp.UserFill and (pf > 0.0) and (Ufill > 0.00001)
1900       then begin
1901              {cable investment is adjusted base on the ratio of the actual and user fill factors)
1902              C_Inv := C_Inv * ((Demand * Length / pf) / UFill);
1903              {Capacity pair feet is calculated base on the user fill factor}
1904              pf    := (Demand * Length) / UFill;
1905            end;
1906
```

```
1907      Pair_Ft_Cap:= Pair_Ft_Cap + pf;  // pair feet of cable placed (pair_ft_cap)
1908      Pair_Ft_Utl:= Pair_Ft_Utl + Demand * Length; // pair feet of cable utilized (pair_ft_utl)
1909
1910      {calculate the returned aerial fiber cable investment value using the cable
1911      and splicing investment (C_Inv) and the strand and placing investment (Inv) }
1912      Result := Inv + C_Inv;
1913      WeAreWatching.LogEvent(mFUNC,'Place_Fi_Aerial','  ('+Clli+','+IntToStr(Xpos)+','+intTostr(Ypos)+
1914                      ','+IntToStr(Length)+','+IntToStr(strands)+','+IntToStr(demand)+
1915                      ','+FloatToStr(UFill)+'='+FloatToStr(Result));
1916  end;        // END Place_FI_Aerial function
1917
1918  Function Place_FI_Buried(Clli : String; ArcType, Xpos, Ypos : integer;
1919                                  Length, Strands, Demand : Longint; UFill : Double) : Double;
1920
1921   { This subroutine calculates the investments for placing buried fiber cables.
1922
1923        Labor activity codes:  (used in Place_FI_Buried)
1924              LP49B:          Place additional cable/duct
1925              LP73A:      Splice Pit
1926              LS50A:      Splice Fiber Optic cable - 48 fibers or less
1927              LS50B:          Splice Fiber Optic cable - greater than 48 fibers
1928   }
1929
1930    Var
1931      Inv, C_Inv, rC_Inv, pf : Double;
1932      CS, tmp_Strands : LongInt;
1933      Total_Fi_Strands : Longint;
1934
1935    begin                     // set initial values
1936      Inv       := 0.0;       // strand and placing investment
1937      C_Inv     := 0.0;       // cable and splicing investment
1938      rC_Inv    := 0.0;       // investment for less than max size cables
1939      pf        := 0.0;       // pair feet
1940      Cable_Cnt:= 0;          // number of cables
1941      Total_Fi_Strands := 0;  // number of fiber strands
1942
1943      Case ArcType of
1944         0 : OSPType := 'I';
1945         1 : OSPType := 'F';
```

```
1946        2,3 : begin
1947                if DM_ICM.t_Arcs.FieldByName('OSP Type').AsString = 'D'
1948                  then OSPType := 'D'
1949                  else OSPType := 'F';
1950              end;
1951          4 : OSPType := 'D';
1952      end;
1953
1954      {test to see if there is a demand and length for buried fiber placement}
1955      if (Demand < 1) or (Length < 1) then begin
1956        Result := 0;
1957        WeAreWatching.LogEvent(mFUNC,'Place_Fi_Buried','  ('+Clli+','+IntToStr(Xpos)+','+intTostr(Ypos)+
1958                        ','+IntToStr(Length)+','+IntToStr(strands)+','+IntToStr(demand)+
1959                        ','+FloatToStr(UFill)+'=0 (Demand<1, Length<1)');
1960        exit;    // if not, then exit
1961      end;
1962
1963      tmp_Strands := Strands;
1964      CS := f_OSPMaterial.MaxSize('FiBur');  // largest buried fiber cable size in material table
1965      if CS > 0           // check for max cable found as precaution
1966        then begin
1967               {calculate the number of cables required to meet the temporary
1968               demand}
1969               cable_cnt:= Trunc(tmp_Strands/CS);
1970               if cable_cnt > 0  // are there any max size cables required?
1971                 then begin
1972                        {At least one max size cable is required so calculate
1973                        investment using the max size cable count.  Write the
1974                        information on the buried fiber cable to the inventory
1975                        file (when a full size cable is placed the segment
1976                        number is written as a negative to the inventory file.
1977                        A positive segment number is used for the smaller cables.) }
1978                        C_Inv  := f_OSPMaterial.Mat('FiBur', CS) *
1979                                  Length * cable_cnt;
1980                        tmp_Strands := tmp_Strands - round(cable_cnt * CS);
1981                        Total_Fi_Strands := Total_Fi_Strands + round(cable_cnt * CS);
1982                        PartInventory.Write (Clli, OSPType[1], 'BURIEDFI', CS, Cable_Cnt*Length, Xpos, Ypos);
1983                      end;
1984             end
```

```
1985        else begin
1986               {no max size cables were found in the material file, so display
1987               an error message }
1988               MessageDlg('Fiber Buried not available', mterror, [mbOk], 0);
1989               Result := 0.0;
1990               WeAreWatching.LogEvent(mERROR,'Place_Fi_Buried','***'+Clli+' Fiber Buried not available');
1991               exit;
1992             end;
1993
1994       {Find cable size for demand or remaining demand }
1995       if tmp_Strands > 0
1996         then begin
1997                CS := f_OSPMaterial.FindSize('FiBur', tmp_Strands);
1998                Inc (Cable_Cnt);
1999                rC_Inv := f_OSPMaterial.Mat('FiBur', CS) * Length;
2000                Total_Fi_Strands := Total_Fi_Strands + CS;
2001                PartInventory.Write (Clli, OSPType[1], 'BURIEDFI', CS, Length, Xpos, Ypos);
2002              end;
2003      {calculate total (material) investment using investment for max size
2004      cables and the investment for less than max size cables }
2005      C_Inv := C_Inv + rC_Inv;
2006
2007     Splice_Pts := Round(length/Asmp.Bur_Fi_Span);
2008     if Splice_Pts = 0 then Splice_Pts := 1;
2009
2010      {calculate the investment with splice pits added since cable being
2011      processed is not distribution cable }
2012       // USF Coding Doc 3/25/98
2013      Inv := Inv + f_OSPLabor.Labor('LP73A',  Tier_Flag)* Splice_Pts;
2014
2015     // USF Coding Doc 3/25/98
2016    if ArcType < 2
2017      then begin
2018             {calculate the splicing investment using labor dependant on the
2019             total number of fiber strands }
2020             if Total_Fi_Strands < 49
2021               then C_Inv := C_Inv + f_OSPLabor.Labor('LS50A', Tier_Flag) * Total_Fi_Strands * Splice_Pts
2022               else C_Inv := C_Inv + f_OSPLabor.Labor('LS50B', Tier_Flag) * Total_Fi_Strands * Splice_Pts;
2023          end;
```

```
2024    {--------------------}
2025
2026      pf := (Demand * Length) /(Strands/Total_Fi_Strands);
2027
2028      {if user wants to use a specific fill factor }
2029      if Asmp.UserFill and (pf > 0.0) and (Ufill > 0.00001)
2030        then begin
2031               {cable investment is adjusted base on the ratio of the actual and user fill factors}
2032               C_Inv := C_Inv * ((Demand * Length / pf) / UFill);
2033               {Capacity pair feet is calculated base on the user fill factor}
2034               pf    := (Demand * Length) / UFill;
2035             end;
2036
2037      if Cable_Cnt > 1 // USF Coding Doc 3/25/98
2038        {more than one cable is being placed in a trench so add to the
2039        investment the labor for placing additional cables }
2040        then C_Inv := C_Inv + f_OSPLabor.Labor('LP49B',Tier_Flag) * (Cable_Cnt-1) * Length;
2041
2042      Pair_Ft_Cap:= Pair_Ft_Cap + pf; // pair feet of cable placed (pair_ft_cap)
2043      Pair_Ft_Utl:= Pair_Ft_Utl + Demand * Length;   // pair feet of cable utilized (pair_ft_utl)
2044
2045      {calculate the returned buried fiber cable investment value using the cable
2046      and splicing investment (C_Inv) and the strand and placing investment (Inv) }
2047      Result := Inv + C_Inv;
2048  //    WeAreWatching.LogEvent(mFUNC,'Place_Fi_Buried','  ('+Clli+','+IntToStr(Xpos)+','+intTostr(Ypos)+
2049  //                  ','+intToStr(segment)+','+IntToStr(Length)+','+IntToStr(strands)+','+IntToStr(demand)+
2050  //                  ','+FloatToStr(UFill)+'='+FloatToStr(Result));
2051
2052   end;    // END Place_FI_Buried function
2053
2054  Function Place_FI_UndGnd(Clli : String; ArcType, Xpos, Ypos : integer;
2055                                  Length, Strands, Demand : Longint; UFill : Double) : Double;
2056
2057   { This subroutine calculates the investments for placing underground fiber cables.
2058
2059        Labor activity codes:  (used in Place_FI_UndGnd)
2060              LP43C:          Place fiber cable in conduit
2061              LS50A:      Splice Fiber Optic cable - 48 fibers or less
2062              LS50B:          Splice Fiber Optic cable - greater than 48 fibers
```

```
2063              LS72A:      Manhole setup
2064    }
2065
2066     Var
2067       Inv, C_Inv, rC_Inv, pf : Double;
2068       CS, tmp_Strands : LongInt;
2069       Total_Fi_Strands : Longint;
2070
2071     begin                        // set initial values
2072       Inv       := 0.0;          // investment for splicing and cable
2073       C_Inv     := 0.0;          // investment for placing and strand
2074       rC_Inv    := 0.0;          // investment for less than max size cables
2075       pf        := 0.0;          // pair feet
2076       Cable_Cnt:= 0;             // number of cables
2077       Total_Fi_Strands := 0;     // number of fiber strands
2078
2079       Case ArcType of
2080          0 : OSPType := 'I';
2081          1 : OSPType := 'F';
2082        2,3 : begin
2083                if DM_ICM.t_Arcs.FieldByName('OSP Type').AsString = 'D'
2084                  then OSPType := 'D'
2085                  else OSPType := 'F';
2086              end;
2087          4 : OSPType := 'D';
2088       end;
2089
2090       {test to see if there is a demand and length for the underground
2091       fiber placement}
2092       if (Demand < 1) or (Length < 1) then begin
2093         Result := 0;
2094         WeAreWatching.LogEvent(mFUNC,'Place_Fi_UndGnd','  ('+Clli+','+IntToStr(Xpos)+','+intTostr(Ypos)+
2095                           ','+IntToStr(Length)+','+IntToStr(strands)+','+IntToStr(demand)+
2096                           ','+FloatToStr(UFill)+'=0 (Demand<1, Length<1)');
2097         exit;    // if not, then exit
2098       end;
2099
2100       tmp_Strands := Strands;
2101       CS := f_OSPMaterial.MaxSize('FiUnd');  // largest underground fiber cable size in material table
```

```
2102      if CS > 0           // check for max cable found as precaution
2103        then begin
2104               {calculate the number of cable required to meet the temporary
2105               demand}
2106               cable_cnt:= Trunc(tmp_Strands/CS);
2107               if cable_cnt > 0  // are there any max size cables required?
2108                 then begin
2109                        {At least one max size cable is required so calculate
2110                        investment using the max size cable count.  Write the
2111                        information on the underground fiber cable to the
2112                        inventory file (when a full size cable is placed the
2113                        segment number is written as a negative to the inventory
2114                        file.  A positive segment number is used for the smaller
2115                        cables.) }
2116                        C_Inv      := f_OSPMaterial.Mat('FiUnd', CS) *
2117                                      Length * cable_cnt;
2118                        tmp_Strands := tmp_Strands - round(cable_cnt * CS);
2119                        Total_Fi_Strands := Total_Fi_Strands + round(cable_cnt * CS);
2120                        PartInventory.Write (Clli, OSPType[1], 'UNDGNDFI', CS, Cable_Cnt*Length, Xpos, Ypos);
2121                      end;
2122             end
2123        else begin
2124               {no max size cables were found in the material file, so display
2125               an error message }
2126               MessageDlg('Fiber undergound not available', mterror, [mbOk], 0);
2127               Result := 0.0;
2128               WeAreWatching.LogEvent(mERROR,'Place_Fi_UndGnd','***'+Clli+' Fiber underground not
2129   available');
2130               exit;
2131             end;
2132
2133       {Find cable size for demand or remaining demand }
2134       if tmp_Strands > 0
2135         then begin
2136                CS     := f_OSPMaterial.FindSize('FiUnd', tmp_Strands);
2137                Inc (Cable_Cnt);
2138                rC_Inv  := f_OSPMaterial.Mat('FiUnd', CS) * Length;
2139                Total_Fi_Strands := Total_Fi_Strands + CS;
2140                PartInventory.Write (Clli, OSPType[1], 'UNDGNDFI', CS, Length, Xpos, Ypos);
```

```
2141              end;
2142        {calculate total (material) investment using investment for max size
2143        cables and the investment for less than max size cables }
2144        C_Inv := C_Inv + rC_Inv;
2145
2146        {calculate the investment using labor for placing fiber in conduit }
2147        // USF Coding Doc 3/25/98
2148        Inv := Inv + f_OSPLabor.Labor('LP43C',  Tier_Flag) * Length * Cable_cnt;
2149        {calculate the investment using labor for manhole site setup since
2150        underground cable is being placed in feeder }
2151
2152        Splice_Pts := Round(length/Asmp.Manhole_Spacing);
2153        if Splice_Pts = 0 then Splice_Pts := 1;
2154
2155        Inv := Inv + f_OSPLabor.Labor('LS72A',  Tier_Flag)* Splice_Pts;
2156        if ArcType < 2
2157          then begin
2158                 {calculate the splicing investment using labor dependant on the
2159                 total number of fiber strands }
2160                 if Total_Fi_Strands < 49
2161                   then C_Inv := C_Inv + f_OSPLabor.Labor('LS50A', Tier_Flag) * Total_Fi_Strands * Splice_Pts
2162                   else C_Inv := C_Inv + f_OSPLabor.Labor('LS50B', Tier_Flag) * Total_Fi_Strands * Splice_Pts;
2163               end;
2164     {--------------------}
2165
2166        pf := (Demand * Length) /(Strands/Total_Fi_Strands);
2167
2168       {if user wants to use a specific fill factor }
2169       if Asmp.UserFill and (pf > 0.0) and (Ufill > 0.00001)
2170          then begin
2171                 {cable investment is adjusted base on the ratio of the actual and user fill factors}
2172                 C_Inv := C_Inv * ((Demand * Length / pf) / UFill);
2173                 {Capacity pair feet is calculated base on the user fill factor}
2174                 pf    := (Demand * Length) / UFill;
2175               end;
2176
2177       Pair_Ft_Cap:= Pair_Ft_Cap + pf; // pair feet of cable placed (pair_ft_cap)
2178       Pair_Ft_Utl:= Pair_Ft_Utl + Demand * Length;   // pair feet of cable utilized (pair_ft_utl)
2179
```

```
2180      {calculate the returned underground fiber cable investment value using
2181      the cable and splicing investment (C_Inv) and the strand and placing
2182      investment (Inv) }
2183      Result  := Inv + C_Inv;
2184  //    WeAreWatching.LogEvent(mFUNC,'Place_Fi_UndGnd',' ('+Clli+','+IntToStr(Xpos)+','+intTostr(Ypos)+
2185  //                  ','+intToStr(segment)+','+IntToStr(Length)+','+IntToStr(strands)+','+IntToStr(demand)+
2186  //                  ','+FloatToStr(UFill)+'='+FloatToStr(Result));
2187    end;        // END of Place_FI_UndGnd function
2188
2189  procedure ArcInvestments;
2190   var
2191     CLLI    : String;
2192     Length  : Integer;
2193     Strands : Integer;
2194     IOF_Stnd: Integer;
2195     HRF_Stnd: Integer;
2196     IRF_Stnd: Integer;  // RFPC0067
2197     Demand  : Integer;
2198     AT      : Integer;  // Arc Type
2199     DPrFtU  : Double;   // Distribution pair feet utilize
2200     DPrFtC  : Double;   // Distribution pair feet capacity
2201     FPrFtU  : Double;   // Feeder pair feet utilize
2202     FPrFtC  : Double;   // Feeder pair feet capacity
2203     UFill   : Double;   // User Fill
2204     CInv    : Double;   //Cable investment
2205     SInv    : Double;   //Structure investment
2206     VS      : Double;   //VS investment
2207     IOF_CInv: Double;   //Inter-Office Cable investment
2208     IOF_SInv: Double;   //Inter-Office Structure investment
2209     IOF_VS  : Double;   //Inter-Office VS investment
2210     HRF_CInv: Double;   //Host-Remote Cable investment
2211     HRF_SInv: Double;   //Host-Remote Structure investment
2212     HRF_VS  : Double;   //Host-Remote VS investment
2213     IRF_CInv: Double;   //InterRing Cable investment
2214     IRF_SInv: Double;   //InterRing Structure investment
2215     IRF_VS  : Double;   //InterRing VS investment
2216     RF      : Double;   //Res lines fraction
2217     BF      : Double;   //Bus lines fraction
2218     DF      : String[1];//Distribution / feeder
```

```
2219     Distance: Double;   //length of arc in miles
2220
2221   function IsRemoteSwitch : Boolean;
2222    Var
2223     SW_Type : String;
2224     TabName : String;
2225
2226   { function IsRemoteSwitch gets a switch type from the Nodes Table (t_nodes)
2227     and determines if it a remote switch by looking it up in the switch type
2228     table (t_switch).  Sets variable OK = False if NOT found. }
2229    begin
2230     { Open Switch Type Table }
2231     TabName :=  f_associate.GetFileName(2001); // get switch type table filename and location using index
2232   2001
2233     With dm_ICM.t_Switch do   // set switch table variable name to t_switch and setup table
2234       begin
2235         if Active then Close;
2236         DatabaseName := ExtractFilePath(TabName);
2237         TableName := ExtractFilename(TabName);
2238         IndexFieldNames := '';
2239         IndexName := '';
2240         TableType := ttDefault;
2241         Exclusive := true;
2242         Open;
2243       end;  // end 'with t_switch do'
2244
2245     SW_Type := dm_ICM.t_Nodes.FieldByName('Switch Type').AsString;  // get switch type from nodes table
2246
2247     if dm_ICM.t_Switch.FindKey([SW_Type])  // search for the nodes switch type in the switch type table
2248       then Result := UpCase(dm_ICM.t_Switch.FieldByName('Host/Rem').AsString[1]) in ['R','D']     //RFPC0060
2249       else begin  // nodes switch type not found in switch type table, return False
2250              MessageDlg(CLLI+' - Could not find switch type ['+SW_Type+']', mtError, [mbOk],0);   //RFPC0060
2251              WeAreWatching.LogEvent(4,'Switch','Could not find switch type ['+SW_Type+']');       //RFPC0060
2252              Result := False;                                                                     //RFPC0060
2253            end;
2254
2255     dm_ICM.t_Switch.Close;
2256   end;  // end IsRemoteSwitch
2257
```

```
2258  begin
2259    if Asmp.CU_Loop_Length > 12
2260      then CUGA := 24   //RFPC0068 use heavier wire for 18KFT and ADSL loops
2261      else CUGA := 26;
2262
2263    DM_ICM.t_HRArcFile.Open;
2264    DM_ICM.t_Arcs.IndexName := '';       //Order Arc table for investment calculations
2265    DM_ICM.t_Nodes.First;                 // Move to first record of Node table
2266    while Not DM_ICM.t_Nodes.Eof do       // Loop through Node table one record at a time
2267       begin
2268         f_Demand.Gauge1.Progress := Round((DM_ICM.t_Nodes.RecNo-1) / DM_ICM.t_Nodes.RecordCount * 100.0);
2269         f_Demand.l_Progress.Caption := Format('Investment %d of %d',
2270                                        [DM_ICM.t_Nodes.RecNo-1, DM_ICM.t_Nodes.RecordCount]);
2271         Application.ProcessMessages;
2272
2273         CLLI     := DM_ICM.t_Nodes.FieldByName('CLLI').AsString; // Set CLLI code
2274         Asmp.HML := DM_ICM.t_Nodes.FieldByName('HML').AsString;  // Set High Medium Low code
2275         DM_ICM.t_Arcs.SetRange([CLLI],[CLLI]);                   // Set filter for CLLI
2276
2277         DPrFtU  :=0.0;  // Distribution pair feet utilize
2278         DPrFtC  :=0.0;  // Distribution pair feet capacity
2279         FPrFtU  :=0.0;  // Feeder pair feet utilize
2280         FPrFtC  :=0.0;  // Feeder pair feet capacity
2281
2282         DM_ICM.t_Arcs.First;              // Move to first record of the ARC table
2283         while Not DM_ICM.t_Arcs.Eof do    // Loop through the arc table one record at a time
2284           begin
2285             BedRock             := DM_ICM.t_Arcs.FieldByName('BedRock').AsInteger;
2286             WaterTable          := DM_ICM.t_Arcs.FieldByName('WaterTable').AsInteger;
2287             Asmp.Structure_Flag := DM_ICM.t_Arcs.FieldByName('Structure').AsInteger;
2288
2289
2290             AT := DM_ICM.t_Arcs.FieldByName('Arc Type').AsInteger;
2291             if AT > 0
2292               then begin
2293                      Demand := DM_ICM.t_Arcs.FieldByName('Res Lines').AsInteger +
2294  DM_ICM.t_Arcs.FieldByName('Bus Lines').AsInteger;
2295                      RF := DM_ICM.t_Arcs.FieldByName('Res Lines').AsFloat / Demand;
2296                      BF := 1.0 - RF;
```

```
2297                      AvgLpLength.Write(Clli, DM_ICM.t_Arcs.FieldByName('LoopLenCode').AsInteger);
2298                    end;
2299
2300              case AT  of
2301                  1 : DF := 'F';
2302                2,3 : DF := DM_ICM.t_Arcs.FieldByName('OSP Type').AsString;
2303                  4 : DF := 'D';
2304              end;
2305
2306              case AT of
2307              2..4 : begin // Distribution copper cable
2308                       if Asmp.UserFill
2309                         then UFill := Asmp.UserDistFill;
2310
2311                       Pair_Ft_Cap := 0;    // pair feet of cable placed (pair_ft_cap)
2312                       Pair_Ft_Utl := 0;   // pair feet of cable utilized (pair_ft_utl)
2313                       FiberCable  := False;
2314                       CanPlow     := DM_ICM.t_Arcs.FieldByName('CanPlow').AsInteger <> 0;
2315
2316                       // Aerial Cable
2317                       Length := Round(DM_ICM.t_Arcs.FieldByName('Length').AsFloat *
2318                                       DM_ICM.t_Arcs.FieldByName('PerAerial').AsFloat);
2319                       if Length > 0
2320                         then begin
2321                                CInv := Place_CU_Aerial(Clli, AT, DM_ICM.t_Arcs.FieldByName('X').AsInteger,
2322                                                                  DM_ICM.t_Arcs.FieldByName('Y').AsInteger,
2323                                                        Length, Demand, UFill);
2324                                SInv := Poles (Clli, AT, DM_ICM.t_Arcs.FieldByName('X').AsInteger,
2325                                                        DM_ICM.t_Arcs.FieldByName('Y').AsInteger, Length);
2326
2327                                if (DM_ICM.t_Arcs.FieldByName('X').AsInteger = 0) and
2328                                   (DM_ICM.t_Arcs.FieldByName('Y').AsInteger = 0)
2329                                  then begin  // Core section of the office
2330                                         OSP_Table.Write (CLLI, '242110',  'T'+DF+'CAERCU', DF, CInv, CInv,
2331  0.0);
2332                                         OSP_Table.Write (CLLI, '242110',  'R'+DF+'CAERCU', DF, CInv * RF,
2333  CInv * RF, 0.0);
2334                                         OSP_Table.Write (CLLI, '242110',  'B'+DF+'CAERCU', DF, CInv * BF,
2335  CInv * BF, 0.0);
```

```
2336                                    OSP_Table.Write (CLLI, '241100',  'T'+DF+'CPOLES', DF, SInv, SInv,
2337   0.0);
2338                                    OSP_Table.Write (CLLI, '241100',  'R'+DF+'CPOLES', DF, SInv * RF,
2339   SInv * RF, 0.0);
2340                                    OSP_Table.Write (CLLI, '241100',  'B'+DF+'CPOLES', DF, SInv * BF,
2341   SInv * BF, 0.0);
2342                                 end
2343                              else begin  // Non-Core section of the office
2344                                    OSP_Table.Write (CLLI, '242110',  'T'+DF+'FAERCU', DF, CInv, CInv,
2345   0.0);
2346                                    OSP_Table.Write (CLLI, '242110',  'R'+DF+'FAERCU', DF, CInv * RF,
2347   CInv * RF, 0.0);
2348                                    OSP_Table.Write (CLLI, '242110',  'B'+DF+'FAERCU', DF, CInv * BF,
2349   CInv * BF, 0.0);
2350                                    OSP_Table.Write (CLLI, '241100',  'T'+DF+'FPOLES', DF, SInv, SInv,
2351   0.0);
2352                                    OSP_Table.Write (CLLI, '241100',  'R'+DF+'FPOLES', DF, SInv * RF,
2353   SInv * RF, 0.0);
2354                                    OSP_Table.Write (CLLI, '241100',  'B'+DF+'FPOLES', DF, SInv * BF,
2355   SInv * BF, 0.0);
2356                                 end;
2357                           end;
2358
2359                        // Buried Cable
2360                        Length := Round(DM_ICM.t_Arcs.FieldByName('Length').AsFloat *
2361                                 DM_ICM.t_Arcs.FieldByName('PerBuried').AsFloat);
2362                        if Length > 0
2363                          then begin
2364                                 CInv := Place_CU_Buried(Clli, AT, DM_ICM.t_Arcs.FieldByName('X').AsInteger,
2365                                                                   DM_ICM.t_Arcs.FieldByName('Y').AsInteger,
2366                                                          Length, Demand, UFill);
2367                                 SInv := Trench (Clli, AT, DM_ICM.t_Arcs.FieldByName('X').AsInteger,
2368                                                          DM_ICM.t_Arcs.FieldByName('Y').AsInteger, Length);
2369
2370                                 if (DM_ICM.t_Arcs.FieldByName('X').AsInteger = 0) and
2371                                    (DM_ICM.t_Arcs.FieldByName('Y').AsInteger = 0)
2372                                   then begin  // Core section of the office
2373                                          OSP_Table.Write (CLLI, '242310',  'T'+DF+'CBURCU', DF,
2374                                                          (CInv+SInv), CInv, SInv);
```

```
2375                                    OSP_Table.Write (CLLI, '242310',  'R'+DF+'CBURCU', DF,
2376                                                     (CInv+SInv) * RF, CInv * RF, SInv * RF);
2377                                    OSP_Table.Write (CLLI, '242310',  'B'+DF+'CBURCU', DF,
2378                                                     (CInv+SInv) * BF, CInv * BF, SInv * BF);
2379                                 end
2380                               else begin  // Non-Core section of the office
2381                                    OSP_Table.Write (CLLI, '242310',  'T'+DF+'FBURCU', DF,
2382                                                     (CInv+SInv), CInv, SInv);
2383                                    OSP_Table.Write (CLLI, '242310',  'R'+DF+'FBURCU', DF,
2384                                                     (CInv+SInv) * RF, CInv * RF, SInv * RF);
2385                                    OSP_Table.Write (CLLI, '242310',  'B'+DF+'FBURCU', DF,
2386                                                     (CInv+SInv) * BF, CInv * BF, SInv * BF);
2387                                 end;
2388                            end;
2389
2390                         //Underground Cable
2391                         Length := Round(DM_ICM.t_Arcs.FieldByName('Length').AsFloat *
2392                                         DM_ICM.t_Arcs.FieldByName('PerUndGnd').AsFloat);
2393                         if Length > 0
2394                           then begin
2395                                  CInv := Place_CU_UndGnd(Clli, AT, DM_ICM.t_Arcs.FieldByName('X').AsInteger,
2396                                                                    DM_ICM.t_Arcs.FieldByName('Y').AsInteger,
2397                                                           Length, Demand, UFill);
2398                                  Num_Of_Ducts := Cable_Cnt;
2399                                  SInv := Conduit (Clli, AT, DM_ICM.t_Arcs.FieldByName('X').AsInteger,
2400                                                             DM_ICM.t_Arcs.FieldByName('Y').AsInteger,
2401                                                             DM_ICM.t_Arcs.FieldByName('Demand').AsInteger,
2402                                                             DM_ICM.t_Arcs.FieldByName('Bus Lines').AsInteger,
2403  Length, 'AVG');
2404                                  VS   := Conduit (Clli, AT, DM_ICM.t_Arcs.FieldByName('X').AsInteger,
2405                                                             DM_ICM.t_Arcs.FieldByName('Y').AsInteger,
2406                                                             DM_ICM.t_Arcs.FieldByName('Demand').AsInteger,
2407                                                             DM_ICM.t_Arcs.FieldByName('Bus Lines').AsInteger,
2408  Length, 'VS');
2409                                  if (DM_ICM.t_Arcs.FieldByName('X').AsInteger = 0) and
2410                                     (DM_ICM.t_Arcs.FieldByName('Y').AsInteger = 0)
2411                                    then begin  // Core section of the office
2412                                           OSP_Table.Write (CLLI, '242210',  'T'+DF+'CUNDCU', DF, CInv, CInv,
2413  0.0);
```

```
2414                                        OSP_Table.Write (CLLI, '242210',  'R'+DF+'CUNDCU', DF, CInv * RF,
2415   CInv * RF, 0.0);
2416                                        OSP_Table.Write (CLLI, '242210',  'B'+DF+'CUNDCU', DF, CInv * BF,
2417   CInv * BF, 0.0);
2418                                        OSP_Table.Write (CLLI, '244100',  'T'+DF+'CCOND',  DF, SInv, VS,
2419   (SInv - VS));
2420                                        OSP_Table.Write (CLLI, '244100',  'R'+DF+'CCOND',  DF, SInv * RF, VS
2421   * RF,
2422                                                         (SInv - VS) * RF);
2423                                        OSP_Table.Write (CLLI, '244100',  'B'+DF+'CCOND',  DF, SInv * BF, VS
2424   * BF,
2425                                                         (SInv - VS) * BF);
2426                                    end
2427                                 else begin  // Non-Core section of the office
2428                                        OSP_Table.Write (CLLI, '242210',  'T'+DF+'FUNDCU', DF, CInv, CInv,
2429   0.0);
2430                                        OSP_Table.Write (CLLI, '242210',  'R'+DF+'FUNDCU', DF, CInv * RF,
2431   CInv * RF, 0.0);
2432                                        OSP_Table.Write (CLLI, '242210',  'B'+DF+'FUNDCU', DF, CInv * BF,
2433   CInv * BF, 0.0);
2434                                        OSP_Table.Write (CLLI, '244100',  'T'+DF+'FCOND',  DF, SInv, VS,
2435   (SInv - VS));
2436                                        OSP_Table.Write (CLLI, '244100',  'R'+DF+'FCOND',  DF, SInv * RF, VS
2437   * RF,
2438                                                         (SInv - VS) * RF);
2439                                        OSP_Table.Write (CLLI, '244100',  'B'+DF+'FCOND',  DF, SInv * BF, VS
2440   * BF,
2441                                                         (SInv - VS) * BF);
2442                                    end;
2443                             end;
2444
2445                       if DF = 'D'
2446                         then begin
2447                                DPrFtU := DPrFtU + Pair_Ft_Utl;  // Distribution pair feet utilize
2448                                DPrFtC := DPrFtC + Pair_Ft_Cap;  // Distribution pair feet capacity
2449                              end
2450                         else begin
2451                                FPrFtU := FPrFtU + Pair_Ft_Utl;  // Feeder pair feet utilize
2452                                FPrFtC := FPrFtC + Pair_Ft_Cap;  // Feeder pair feet capacity
```

```
2453                        end;
2454                    end; //end of backbone cables
2455
2456                1 : begin  //Feeder, IOF fiber cables
2457                       if Asmp.UserFill
2458                         then UFill := Asmp.UserFeedFill;
2459                       FiberCable := True;
2460                       if Asmp.HML = 'L'
2461                         then Canplow := True
2462                         else Canplow := False;
2463
2464                       Pair_Ft_Cap := 0;    // pair feet of cable placed (pair_ft_cap)
2465                       Pair_Ft_Utl := 0;  // pair feet of cable utilized (pair_ft_utl)
2466
2467                       DM_ICM.t_HRArcFile.SetRange([CLLI, AT,
2468                                                    DM_ICM.t_Arcs.FieldByName('From X').AsInteger,
2469                                                    DM_ICM.t_Arcs.FieldByName('From Y').AsInteger,
2470                                                    DM_ICM.t_Arcs.FieldByName('To X').AsInteger,
2471                                                    DM_ICM.t_Arcs.FieldByName('To Y').AsInteger],
2472                                                 [CLLI, AT,
2473                                                    DM_ICM.t_Arcs.FieldByName('From X').AsInteger,
2474                                                    DM_ICM.t_Arcs.FieldByName('From Y').AsInteger,
2475                                                    DM_ICM.t_Arcs.FieldByName('To X').AsInteger,
2476                                                    DM_ICM.t_Arcs.FieldByName('To Y').AsInteger]);
2477
2478
2479                       Strands  := DM_ICM.t_Arcs.FieldByName('Demand').AsInteger +
2480                                   DM_ICM.t_Arcs.FieldByName('IOF').AsInteger +
2481                                   DM_ICM.t_Arcs.FieldByName('HRF').AsInteger +
2482                                   DM_ICM.t_Arcs.FieldByName('IRF').AsInteger;   // RFPC0067
2483                       IOF_Stnd := DM_ICM.t_Arcs.FieldByName('IOF').AsInteger;
2484                       HRF_Stnd := DM_ICM.t_Arcs.FieldByName('HRF').AsInteger;
2485                       IRF_Stnd := DM_ICM.t_Arcs.FieldByName('IRF').AsInteger;   // RFPC0067
2486
2487                       // Aerial Cable
2488                       Length := Round(DM_ICM.t_Arcs.FieldByName('Length').AsFloat *
2489                                       DM_ICM.t_Arcs.FieldByName('PerAerial').AsFloat);
2490                       if Length > 0
2491                         then begin
```

```
2492                            Distance:=Length/5280;
2493                            CInv := Place_FI_Aerial(Clli, AT, DM_ICM.t_Arcs.FieldByName('X').AsInteger,
2494                                                              DM_ICM.t_Arcs.FieldByName('Y').AsInteger,
2495                                                    Length, Strands, Demand, UFill);
2496                            SInv := Poles (Clli, AT, DM_ICM.t_Arcs.FieldByName('X').AsInteger,
2497                                                    DM_ICM.t_Arcs.FieldByName('Y').AsInteger, Length);
2498
2499                            // Inter-Office sharing with feeder (if any)
2500                            IOF_CInv := CInv * (IOF_Stnd / Strands);
2501                            IOF_SInv := SInv * (IOF_Stnd / Strands);
2502                            HRF_CInv := CInv * (HRF_Stnd / Strands);
2503                            HRF_SInv := SInv * (HRF_Stnd / Strands);
2504                            IRF_CInv := CInv * (IRF_Stnd / Strands);
2505                            IRF_SInv := SInv * (IRF_Stnd / Strands);
2506                            CInv     := CInv - (IOF_CInv + HRF_CInv + IRF_CInv);
2507                            SInv     := SInv - (IOF_SInv + HRF_SInv + IRF_SInv);
2508
2509                            OSP_Table.Write (CLLI, '242120', 'TFFAERFI', 'F', CInv, CInv, 0.0);
2510                            OSP_Table.Write (CLLI, '242120', 'RFFAERFI', 'F', CInv * RF, CInv * RF, 0.0);
2511                            OSP_Table.Write (CLLI, '242120', 'BFFAERFI', 'F', CInv * BF, CInv * BF, 0.0);
2512                            OSP_Table.Write (CLLI, '241100', 'TFFPOLEFI','F', SInv, SInv, 0.0);
2513                            OSP_Table.Write (CLLI, '241100', 'RFFPOLEFI','F', SInv * RF, SInv * RF, 0.0);
2514                            OSP_Table.Write (CLLI, '241100', 'BFFPOLEFI','F', SInv * BF, SInv * BF, 0.0);
2515
2516                            if IOF_Stnd > 0
2517                              then begin
2518                                     IOF_Table.Write (CLLI, '242120', 'IHAERFI', 'I', IOF_CInv, 0.0,
2519   IOF_CInv, True);
2520                                     IOF_Table.Write (CLLI, '241100', 'IHPOLES', 'I', IOF_SInv, 0.0,
2521   IOF_SInv, True);
2522                                     // output the distances
2523                                     IOF_Table.Write (CLLI, '000000', 'IHAERDIST', 'I',
2524                                                      Distance, Distance, Distance, False);
2525                                     IOF_Table.Write (CLLI, '000000', 'IHLEN', 'I', Distance,
2526                                                      Distance, Distance, False);
2527                                   end;
2528
2529                            if IRF_Stnd > 0
2530                              then begin   //RFPC0067
```

```
2531                                        IOF_Table.Write (CLLI, '242120', 'IIAERFI', 'I', IRF_CInv, 0.0,
2532  IRF_CInv, False);
2533                                        IOF_Table.Write (CLLI, '241100', 'IIPOLES', 'I', IRF_SInv, 0.0,
2534  IRF_SInv, False);
2535                                        // output the distances
2536                                        IOF_Table.Write (CLLI, '000000', 'IIAERDIST', 'I',
2537                                                         Distance, Distance, Distance, False);
2538                                        IOF_Table.Write (CLLI, '000000', 'IILEN', 'I', Distance,
2539                                                         Distance, Distance, False);
2540                                     end;
2541
2542                                  if HRF_Stnd > 0
2543                                    then begin
2544                                           HRF_CInv := HRF_CInv / (HRF_Stnd / 4.0);
2545                                           HRF_SInv := HRF_SInv / (HRF_Stnd / 4.0);
2546                                           if IsRemoteSwitch
2547                                             then begin
2548                                                    IOF_Table.Write (CLLI, '242120', 'IRAERFI', 'I',
2549                                                                     HRF_CInv, 0.0, HRF_CInv, False);
2550                                                    IOF_Table.Write (CLLI, '241100', 'IRPOLES', 'I',
2551                                                                     HRF_SInv, 0.0, HRF_SInv, False);
2552                                                    // output the distances
2553                                                    IOF_Table.Write (CLLI, '000000', 'IRAERDIST', 'I',
2554                                                                     Distance, Distance, Distance, False);
2555                                                    IOF_Table.Write (CLLI, '000000', 'IHRLEN', 'I',
2556                                                                     Distance, Distance, Distance, False);
2557                                                  end
2558                                             else begin
2559                                                    DM_ICM.t_HRArcFile.First;
2560                                                    while Not DM_ICM.t_HRArcFile.EOF do
2561                                                      begin
2562                                                        IOF_Table.Write
2563  (DM_ICM.t_HRArcFile.FieldByName('RCLLI').AsString,
2564                                                                          '242120', 'IRAERFI', 'I',HRF_CInv, 0.0,
2565  HRF_CInv,
2566                                                                          False);
2567                                                        IOF_Table.Write
2568  (DM_ICM.t_HRArcFile.FieldByName('RCLLI').AsString,
```

```
2569                                                                    '241100', 'IRPOLES', 'I', HRF_SInv,
2570  0.0, HRF_SInv,
2571                                                                     False);
2572                                                        // output the distances
2573                                                        IOF_Table.Write
2574  (DM_ICM.t_HRArcFile.FieldByName('RCLLI').AsString,
2575                                                                   '000000', 'IRAERDIST', 'I',
2576                                                                    Distance, Distance, Distance, False);
2577                                                        IOF_Table.Write
2578  (DM_ICM.t_HRArcFile.FieldByName('RCLLI').AsString,
2579                                                                   '000000', 'IHRLEN', 'I',
2580                                                                    Distance, Distance, Distance, False);
2581                                                        DM_ICM.t_HRArcFile.Next;
2582                                                    end;
2583                                                end;
2584                                        end;
2585                                end;
2586
2587                        // Buried Cable
2588                        Length := Round(DM_ICM.t_Arcs.FieldByName('Length').AsFloat *
2589                                        DM_ICM.t_Arcs.FieldByName('PerBuried').AsFloat);
2590                        if Length > 0
2591                          then begin
2592                                 Distance:=Length/5280;
2593                                 CInv := Place_FI_Buried(Clli, AT, DM_ICM.t_Arcs.FieldByName('X').AsInteger,
2594                                                                   DM_ICM.t_Arcs.FieldByName('Y').AsInteger,
2595                                                          Length, Strands, Demand, UFill);
2596                                 SInv := Trench (Clli, AT, DM_ICM.t_Arcs.FieldByName('X').AsInteger,
2597                                                        DM_ICM.t_Arcs.FieldByName('Y').AsInteger, Length);
2598
2599                                 // Inter-Office sharing with feeder (if any)
2600                                 IOF_CInv := CInv * (IOF_Stnd / Strands);
2601                                 IOF_SInv := SInv * (IOF_Stnd / Strands);
2602                                 HRF_CInv := CInv * (HRF_Stnd / Strands);
2603                                 HRF_SInv := SInv * (HRF_Stnd / Strands);
2604                                 IRF_CInv := CInv * (IRF_Stnd / Strands);  //RFPC0067
2605                                 IRF_SInv := SInv * (IRF_Stnd / Strands);  //RFPC0067
2606                                 CInv     := CInv - (IOF_CInv + HRF_CInv + IRF_CInv);
2607                                 SInv     := SInv - (IOF_SInv + HRF_SInv + IRF_SInv);
```

```
2608
2609                              OSP_Table.Write (CLLI, '242320', 'TFFBURFI', 'F', (CInv+SInv), CInv, SInv);
2610                              OSP_Table.Write (CLLI, '242320', 'RFFBURFI', 'F', (CInv+SInv)*RF, CInv*RF,
2611   SInv*RF);
2612                              OSP_Table.Write (CLLI, '242320', 'BFFBURFI', 'F', (CInv+SInv)*BF, CInv*BF,
2613   SInv*BF);
2614
2615                              if IOF_Stnd > 0
2616                                then begin
2617                                       IOF_Table.Write (CLLI, '242320', 'IHBURFI', 'I',
2618                                                        (IOF_CInv+IOF_SInv), 0.0, (IOF_CInv+IOF_SInv), True);
2619                                       // output the distances
2620                                       IOF_Table.Write (CLLI, '000000', 'IHBURDIST', 'I',
2621                                                        Distance, Distance, Distance, False);
2622                                       IOF_Table.Write (CLLI, '000000', 'IHLEN', 'I',
2623                                                        Distance, Distance, Distance, False);
2624                                     end;
2625
2626                              if IRF_Stnd > 0
2627                                then begin  // RFPC0067
2628                                       IOF_Table.Write (CLLI, '242320', 'IIBURFI', 'I',
2629                                                        (IRF_CInv+IRF_SInv), 0.0, (IRF_CInv+IRF_SInv),
2630   False);
2631                                       // output the distances
2632                                       IOF_Table.Write (CLLI, '000000', 'IIBURDIST', 'I',
2633                                                        Distance, Distance, Distance, False);
2634                                       IOF_Table.Write (CLLI, '000000', 'IILEN', 'I',
2635                                                        Distance, Distance, Distance, False);
2636                                     end;
2637
2638                              if HRF_Stnd > 0
2639                                then begin
2640                                       HRF_CInv := HRF_CInv / (HRF_Stnd / 4.0);
2641                                       HRF_SInv := HRF_SInv / (HRF_Stnd / 4.0);
2642                                       if IsRemoteSwitch
2643                                         then begin
2644                                                IOF_Table.Write (CLLI, '242320', 'IRBURFI', 'I',
2645                                                                 (HRF_CInv+HRF_SInv), 0.0,
2646   (HRF_CInv+HRF_SInv), False);
```

```
2647                                                    // output the distances
2648                                                    IOF_Table.Write (CLLI, '000000', 'IRBURDIST', 'I',
2649                                                                     Distance, Distance, Distance, False);
2650                                                    IOF_Table.Write (CLLI, '000000', 'IHRLEN', 'I',
2651                                                                     Distance, Distance, Distance, False);
2652                                                end
2653                                            else begin
2654                                                    DM_ICM.t_HRArcFile.First;
2655                                                    while Not DM_ICM.t_HRArcFile.EOF do
2656                                                      begin
2657                                                        IOF_Table.Write
2658   (DM_ICM.t_HRArcFile.FieldByName('RCLLI').AsString,
2659                                                                     '242320', 'IRBURFI', 'I',
2660   (HRF_CInv+HRF_SInv),
2661                                                                     0.0, (HRF_CInv+HRF_SInv), False);
2662                                                        // output the distances
2663                                                        IOF_Table.Write
2664   (DM_ICM.t_HRArcFile.FieldByName('RCLLI').AsString,
2665                                                                     '000000', 'IRBURDIST', 'I',
2666                                                                     Distance, Distance, Distance, False);
2667                                                        IOF_Table.Write
2668   (DM_ICM.t_HRArcFile.FieldByName('RCLLI').AsString,
2669                                                                     '000000', 'IHRLEN', 'I',
2670                                                                     Distance, Distance, Distance, False);
2671                                                        DM_ICM.t_HRArcFile.Next;
2672                                                end;
2673                                            end;
2674                                        end;
2675                                end;
2676
2677                        //Underground Cable
2678                        Length := Round(DM_ICM.t_Arcs.FieldByName('Length').AsFloat *
2679                                        DM_ICM.t_Arcs.FieldByName('PerUndGnd').AsFloat);
2680                        if Length > 0
2681                          then begin
2682                                 Distance:=Length/5280;
2683                                 CInv := Place_FI_UndGnd(Clli, AT, DM_ICM.t_Arcs.FieldByName('X').AsInteger,
2684                                                                   DM_ICM.t_Arcs.FieldByName('Y').AsInteger,
2685                                                         Length, Strands, Demand, UFill);
```

```
                                Num_Of_Ducts := Ceil(Cable_Cnt / 3.0);
                                SInv := Conduit (Clli, AT, DM_ICM.t_Arcs.FieldByName('X').AsInteger,
                                                           DM_ICM.t_Arcs.FieldByName('Y').AsInteger,
                                                           DM_ICM.t_Arcs.FieldByName('Demand').AsInteger,
                                                           DM_ICM.t_Arcs.FieldByName('Bus Lines').AsInteger,
Length, 'AVG');
                                VS   := Conduit (Clli, AT, DM_ICM.t_Arcs.FieldByName('X').AsInteger,
                                                           DM_ICM.t_Arcs.FieldByName('Y').AsInteger,
                                                           DM_ICM.t_Arcs.FieldByName('Demand').AsInteger,
                                                           DM_ICM.t_Arcs.FieldByName('Bus Lines').AsInteger,
Length, 'VS');

                                // Inter-Office sharing with feeder (if any)
                                IOF_CInv := CInv * (IOF_Stnd / Strands);
                                IOF_SInv := SInv * (IOF_Stnd / Strands);
                                IOF_VS   := VS   * (IOF_Stnd / Strands);
                                HRF_CInv := CInv * (HRF_Stnd / Strands);
                                HRF_SInv := SInv * (HRF_Stnd / Strands);
                                HRF_VS   := VS   * (HRF_Stnd / Strands);
                                IRF_CInv := CInv * (IRF_Stnd / Strands);  //RFPC0067
                                IRF_SInv := SInv * (IRF_Stnd / Strands);  //RFPC0067
                                IRF_VS   := VS   * (IRF_Stnd / Strands);  //RFPC0067
                                CInv     := CInv - (IOF_CInv + HRF_CInv + IRF_CInv);
                                SInv     := SInv - (IOF_SInv + HRF_SInv + IRF_SInv);
                                VS       := VS   - (IOF_VS   + HRF_VS   + IRF_VS);

                                OSP_Table.Write (CLLI, '242220',  'TFFUNDFI', 'F', CInv, CInv, 0.0);
                                OSP_Table.Write (CLLI, '242220',  'RFFUNDFI', 'F', CInv * RF, CInv * RF,
0.0);
                                OSP_Table.Write (CLLI, '242220',  'BFFUNDFI', 'F', CInv * BF, CInv * BF,
0.0);
                                OSP_Table.Write (CLLI, '244100',  'TFFCOND',  'F', SInv, VS, (SInv - VS));
                                OSP_Table.Write (CLLI, '244100',  'RFFCOND',  'F', SInv * RF, VS * RF,(SInv-
VS) * RF);
                                OSP_Table.Write (CLLI, '244100',  'BFFCOND',  'F', SInv * BF, VS * BF,(SInv-
VS) * BF);

                                if IOF_Stnd > 0
                                  then begin
```

```
2725                    IOF_Table.Write (CLLI, '242220', 'IHUNDFI', 'I',
2726                                     IOF_CInv, 0.0, IOF_CInv, True);
2727                    IOF_Table.Write (CLLI, '244100', 'IHCOND',  'I',
2728                                     IOF_SInv, 0.0, IOF_SInv, True);
2729                    // output the distances
2730                    IOF_Table.Write (CLLI, '000000', 'IHUNGDIST', 'I',
2731                                     Distance, Distance, Distance, False);
2732                    IOF_Table.Write (CLLI, '000000', 'IHLEN', 'I',
2733                                     Distance, Distance, Distance, False);
2734                end;
2735
2736            if IRF_Stnd > 0  //RFPC0067
2737              then begin
2738                    IOF_Table.Write (CLLI, '242220', 'IIUNDFI', 'I',
2739                                     IRF_CInv, 0.0, IRF_CInv, False);
2740                    IOF_Table.Write (CLLI, '244100', 'IICOND',  'I',
2741                                     IRF_SInv, 0.0, IRF_SInv, False);
2742                    // output the distances
2743                    IOF_Table.Write (CLLI, '000000', 'IIUNGDIST', 'I',
2744                                     Distance, Distance, Distance, False);
2745                    IOF_Table.Write (CLLI, '000000', 'IILEN', 'I',
2746                                     Distance, Distance, Distance, False);
2747                end;
2748
2749            if HRF_Stnd > 0
2750              then begin
2751                    HRF_CInv := HRF_CInv / (HRF_Stnd / 4.0);
2752                    HRF_SInv := HRF_SInv / (HRF_Stnd / 4.0);
2753                    HRF_VS   := HRF_VS / (HRF_Stnd / 4.0);
2754                    if IsRemoteSwitch
2755                      then begin
2756                            IOF_Table.Write (CLLI, '242220', 'IRUNDFI', 'I',
2757                                             HRF_CInv, 0.0, HRF_CInv, False);
2758                            IOF_Table.Write (CLLI, '244100', 'IRCOND',  'I',
2759                                             HRF_SInv, 0.0, HRF_SInv, False);
2760                            // output the distances
2761                            IOF_Table.Write (CLLI, '000000', 'IRUNGDIST', 'I',
2762                                             Distance, Distance, Distance, False);
2763                            IOF_Table.Write (CLLI, '000000', 'IHRLEN', 'I',
```

```
2764                                                                    Distance, Distance, Distance, False);
2765                                                end
2766                                            else begin
2767                                                   DM_ICM.t_HRArcFile.First;
2768                                                   while Not DM_ICM.t_HRArcFile.EOF do
2769                                                     begin
2770                                                       IOF_Table.Write
2771   (DM_ICM.t_HRArcFile.FieldByName('RCLLI').AsString,
2772                                                                '242220', 'IRUNDFI', 'I', HRF_CInv, 0.0,
2773   HRF_CInv, False);
2774                                                       IOF_Table.Write
2775   (DM_ICM.t_HRArcFile.FieldByName('RCLLI').AsString,
2776                                                                '244100', 'IRCOND',  'I', HRF_SInv, 0.0,
2777   HRF_SInv, False);
2778                                                       // output the distances
2779                                                       IOF_Table.Write
2780   (DM_ICM.t_HRArcFile.FieldByName('RCLLI').AsString,
2781                                                                '000000', 'IRUNGDIST', 'I', Distance,
2782   Distance, Distance, False);
2783                                                       IOF_Table.Write
2784   (DM_ICM.t_HRArcFile.FieldByName('RCLLI').AsString,
2785                                                                '000000', 'IHRLEN', 'I', Distance, Distance,
2786   Distance, False);
2787                                                       DM_ICM.t_HRArcFile.Next;
2788                                                     end;
2789                                                end;
2790                                        end;
2791                                end;
2792
2793                        FPrFtU := FPrFtU + Pair_Ft_Utl;  // Feeder pair feet utilize
2794                        FPrFtC := FPrFtC + Pair_Ft_Cap;  // Feeder pair feet capacity
2795                      end; //end of fiber cables
2796
2797                  0 : begin  //IOF
2798                         FiberCable := True;
2799                         if Asmp.HML = 'L'
2800                           then Canplow := True
2801                           else Canplow := False;
2802
```

```
2803
2804                    Strands  := DM_ICM.t_Arcs.FieldByName('IOF').AsInteger +
2805                                DM_ICM.t_Arcs.FieldByName('HRF').AsInteger +
2806                                DM_ICM.t_Arcs.FieldByName('IRF').AsInteger;
2807                    IOF_Stnd := DM_ICM.t_Arcs.FieldByName('IOF').AsInteger;
2808                    HRF_Stnd := DM_ICM.t_Arcs.FieldByName('HRF').AsInteger;
2809                    IRF_Stnd := DM_ICM.t_Arcs.FieldByName('IRF').AsInteger;  //RFPC0067
2810
2811                    // Aerial Cable
2812                    Length := Round(DM_ICM.t_Arcs.FieldByName('Length').AsFloat *
2813                                    DM_ICM.t_Arcs.FieldByName('PerAerial').AsFloat);
2814                    if Length > 0
2815                      then begin
2816                             Distance:=Length/5280;
2817                             CInv := Place_FI_Aerial(Clli, AT, DM_ICM.t_Arcs.FieldByName('X').AsInteger,
2818                                                               DM_ICM.t_Arcs.FieldByName('Y').AsInteger,
2819                                                     Length, Strands, Demand, UFill);
2820                             SInv := Poles (Clli, AT, DM_ICM.t_Arcs.FieldByName('X').AsInteger,
2821                                                    DM_ICM.t_Arcs.FieldByName('Y').AsInteger, Length);
2822
2823                             if IRF_Stnd > 0  //RFPC0067
2824                               then begin
2825                                      IOF_Table.Write (CLLI, '242120', 'IIAERFI', 'I', CInv, 0.0, CInv,
2826  False);
2827                                      IOF_Table.Write (CLLI, '241100', 'IIPOLES', 'I', SInv, 0.0, SInv,
2828  False);
2829                                      // output the distances
2830                                      IOF_Table.Write (CLLI, '000000', 'IIAERDIST', 'I',
2831                                        Distance, Distance, Distance, False);
2832                                      IOF_Table.Write (CLLI, '000000', 'IILEN', 'I',
2833                                        Distance, Distance, Distance, False);
2834                                    end;
2835
2836                             if IOF_Stnd > 0
2837                               then begin
2838                                      IOF_Table.Write (CLLI, '242120', 'IHAERFI', 'I', CInv, 0.0, CInv,
2839  True);
2840                                      IOF_Table.Write (CLLI, '241100', 'IHPOLES', 'I', SInv, 0.0, SInv,
2841  True);
```

```
2842                                    // output the distances
2843                                    IOF_Table.Write (CLLI, '000000', 'IHAERDIST', 'I',
2844                                      Distance, Distance, Distance, False);
2845                                    IOF_Table.Write (CLLI, '000000', 'IHLEN', 'I',
2846                                      Distance, Distance, Distance, False);
2847                                   end;
2848
2849                                if HRF_Stnd > 0
2850                                  then begin
2851                                    IOF_Table.Write (CLLI, '242120', 'IRAERFI', 'I', CInv, 0.0, CInv,
2852  False);
2853                                    IOF_Table.Write (CLLI, '241100', 'IRPOLES', 'I', SInv, 0.0, SInv,
2854  False);
2855                                    // output the distances
2856                                    IOF_Table.Write (CLLI, '000000','IRAERDIST', 'I',
2857                                      Distance, Distance, Distance, False);
2858                                    IOF_Table.Write (CLLI, '000000','IHRLEN', 'I',
2859                                      Distance, Distance, Distance, False);
2860                                   end;
2861                             end;
2862
2863                        // Buried Cable
2864                        Length := Round(DM_ICM.t_Arcs.FieldByName('Length').AsFloat *
2865                                  DM_ICM.t_Arcs.FieldByName('PerBuried').AsFloat);
2866                        if Length > 0
2867                          then begin
2868                                 Distance:=Length/5280;
2869                                 CInv := Place_FI_Buried(Clli, AT, DM_ICM.t_Arcs.FieldByName('X').AsInteger,
2870                                                                   DM_ICM.t_Arcs.FieldByName('Y').AsInteger,
2871                                                          Length, Strands, Demand, UFill);
2872                                 SInv := Trench (Clli, AT, DM_ICM.t_Arcs.FieldByName('X').AsInteger,
2873                                                         DM_ICM.t_Arcs.FieldByName('Y').AsInteger, Length);
2874
2875                                 if IRF_Stnd > 0   //RFPC0067
2876                                   then begin
2877                                     IOF_Table.Write (CLLI, '242320', 'IIBURFI','I',
2878                                                      (CInv +SInv), 0.0, (CInv +SInv), False);
2879                                     // output the distances
2880                                     IOF_Table.Write (CLLI, '000000', 'IIBURDIST','I',
```

```
2881                                    Distance, Distance, Distance, False);
2882                                  IOF_Table.Write (CLLI, '000000', 'IILEN','I',
2883                                    Distance, Distance, Distance, False);
2884                                end;
2885
2886                              if IOF_Stnd > 0
2887                                then begin
2888                                       IOF_Table.Write (CLLI, '242320', 'IHBURFI','I',
2889                                                        (CInv +SInv), 0.0, (CInv +SInv), True);
2890                                       // output the distances
2891                                       IOF_Table.Write (CLLI, '000000', 'IHBURDIST','I',
2892                                         Distance, Distance, Distance, False);
2893                                       IOF_Table.Write (CLLI, '000000', 'IHLEN','I',
2894                                         Distance, Distance, Distance, False);
2895                                     end;
2896
2897                              if  HRF_Stnd > 0
2898                                then begin
2899                                       IOF_Table.Write (CLLI, '242320', 'IRBURFI','I',
2900                                                       (CInv +SInv), 0.0, (CInv +SInv), False);
2901                                       // output the distances
2902                                       IOF_Table.Write (CLLI, '000000', 'IRBURDIST','I',
2903                                         Distance, Distance, Distance, False);
2904                                       IOF_Table.Write (CLLI, '000000', 'IHRLEN','I',
2905                                         Distance, Distance, Distance, False);
2906                                     end;
2907                            end;
2908
2909                          //Underground Cable
2910                          Length := Round(DM_ICM.t_Arcs.FieldByName('Length').AsFloat *
2911                                     DM_ICM.t_Arcs.FieldByName('PerUndGnd').AsFloat);
2912                          if Length > 0
2913                            then begin
2914                                   Distance:=Length/5280;
2915                                   CInv := Place_FI_UndGnd(Clli, AT, DM_ICM.t_Arcs.FieldByName('X').AsInteger,
2916                                                                     DM_ICM.t_Arcs.FieldByName('Y').AsInteger,
2917                                                                Length, Strands, Demand, UFill);
2918                                   Num_Of_Ducts := Ceil(Cable_Cnt / 3.0);
2919                                   SInv := Conduit (Clli, AT, DM_ICM.t_Arcs.FieldByName('X').AsInteger,
```

```
2920                                                    DM_ICM.t_Arcs.FieldByName('Y').AsInteger,
2921                                                    DM_ICM.t_Arcs.FieldByName('Demand').AsInteger,
2922                                                    DM_ICM.t_Arcs.FieldByName('Bus Lines').AsInteger,
2923   Length, 'AVG');
2924                                VS    := Conduit (Clli, AT, DM_ICM.t_Arcs.FieldByName('X').AsInteger,
2925                                                    DM_ICM.t_Arcs.FieldByName('Y').AsInteger,
2926                                                    DM_ICM.t_Arcs.FieldByName('Demand').AsInteger,
2927                                                    DM_ICM.t_Arcs.FieldByName('Bus Lines').AsInteger,
2928   Length, 'VS');
2929
2930                                if IRF_Stnd > 0  //RFPC0067
2931                                  then begin
2932                                         IOF_Table.Write (CLLI, '242220', 'IIUNDFI', 'I', CInv, 0.0, CInv,
2933   False);
2934                                         IOF_Table.Write (CLLI, '244100', 'IICOND',  'I', SInv, 0.0, SInv,
2935   False);
2936                                         // output the distances
2937                                         IOF_Table.Write (CLLI, '000000', 'IIUNGDIST', 'I',
2938                                           Distance, Distance, Distance, False);
2939                                         IOF_Table.Write (CLLI, '000000', 'IILEN', 'I',
2940                                           Distance, Distance, Distance, False);
2941                                       end;
2942
2943                                if IOF_Stnd > 0
2944                                  then begin
2945                                         IOF_Table.Write (CLLI, '242220', 'IHUNDFI', 'I', CInv, 0.0, CInv,
2946   True);
2947                                         IOF_Table.Write (CLLI, '244100', 'IHCOND',  'I', SInv, 0.0, SInv,
2948   True);
2949                                         // output the distances
2950                                         IOF_Table.Write (CLLI, '000000', 'IHUNGDIST', 'I',
2951                                           Distance, Distance, Distance, False);
2952                                         IOF_Table.Write (CLLI, '000000', 'IHLEN', 'I',
2953                                           Distance, Distance, Distance, False);
2954                                       end;
2955
2956                                if HRF_Stnd > 0
2957                                  then begin
```

```
2958                                        IOF_Table.Write (CLLI, '242220', 'IRUNDFI', 'I', CInv, 0.0, CInv,
2959  False);
2960                                        IOF_Table.Write (CLLI, '244100', 'IRCOND',  'I', SInv, 0.0, SInv,
2961  False);
2962                                        // output the distances
2963                                        IOF_Table.Write (CLLI, '000000', 'IRUNGDIST', 'I',
2964                                          Distance, Distance, Distance, False);
2965                                        IOF_Table.Write (CLLI, '000000', 'IHRLEN', 'I',
2966                                          Distance, Distance, Distance, False);
2967                                      end;
2968                                end;
2969                            end;   //end of IOF
2970                        end;  // end case of AT
2971
2972                        DM_ICM.t_Arcs.Next;
2973                    end;
2974
2975                  // Calculate and save cable utilization for distribution and feeder
2976                  DM_ICM.t_Nodes.Edit;
2977                  if DPrFtC > 0
2978                    then DM_ICM.t_Nodes.FieldByName('DU').AsFloat := DPrFtU / DPrFtC
2979                    else DM_ICM.t_Nodes.FieldByName('DU').AsFloat := 0.0;
2980                  if FPrFtC > 0
2981                    then DM_ICM.t_Nodes.FieldByName('FU').AsFloat := FPrFtU / FPrFtC
2982                    else DM_ICM.t_Nodes.FieldByName('FU').AsFloat := 0.0;
2983                  DM_ICM.t_Nodes.Post;
2984
2985                  DM_ICM.t_Nodes.Next;
2986                end;
2987
2988              DM_ICM.t_HRArcFile.Close;
2989           end;
2990  end.
```

PLACE.pas

```
1
2
3   unit Place;
4
5   interface
6    Procedure Placement;
7
8   implementation
9     Uses
10      ICMDM, OSPTypes, Demand, SysUtils, Forms, dbtables;
11
12  Procedure Placement;
13       //*********************************************************************
14       // This routine determines the arc placement for both feeder and
15       // distribution.
16       //*********************************************************************
17
18   Var
19     q_Arcs           : TQuery;
20     CLLI             : String;
21     Total_Distance   : Integer;
22     IOF_Distance     : Integer;
23     Target_Distance  : Integer;
24     Distance         : Integer;
25     Len              : Integer;
26     PU, PA, PB       : Double;
27     Per_Und          : Double;
28     Per_Bur          : Double;
29     Per_Aer          : Double;
30
31    procedure UpdateArc( U, A, B : Double);
32      begin
33        if DM_ICM.t_Arcs.FindKey([q_Arcs.FieldByName('CLLI').AsString,
34                                  q_Arcs.FieldByName('Arc Type').AsString,
35                                  q_Arcs.FieldByName('From X').AsInteger,
36                                  q_Arcs.FieldByName('From Y').AsInteger,
```

```
                                q_Arcs.FieldByName('To X').AsInteger,
                                q_Arcs.FieldByName('To Y').AsInteger,
                                q_Arcs.FieldByName('Node').AsInteger])
        then begin
               DM_ICM.t_Arcs.Edit;
               DM_ICM.t_Arcs.FieldByName('PerUndGnd').AsFloat := U;
               DM_ICM.t_Arcs.FieldByName('PerAerial').AsFloat := A;
               DM_ICM.t_Arcs.FieldByName('PerBuried').AsFloat := B;
               DM_ICM.t_Arcs.Post;
             end;
    end;

begin
  q_Arcs := TQuery.Create(f_Demand.Owner);            // Create a query

  DM_ICM.t_Nodes.First;                 // Move to first record of Node table
  while Not DM_ICM.t_Nodes.Eof do       // Loop through Node table one record a time
     begin
       f_Demand.Gauge1.Progress := Round((DM_ICM.t_Nodes.RecNo-1) / DM_ICM.t_Nodes.RecordCount * 100.0);
       f_Demand.l_Progress.Caption := Format('Placement %d of %d',
                                     [DM_ICM.t_Nodes.RecNo-1, DM_ICM.t_Nodes.RecordCount]);
       Application.ProcessMessages;

       CLLI    := DM_ICM.t_Nodes.FieldByName('CLLI').AsString; // Set CLLI code

      //*********************************************************************
      //                      FIBER
      //*********************************************************************
      // Setup query for Inter-Office
       q_Arcs.Close;
       q_Arcs.DataBaseName := 'modata';
       q_Arcs.SQL.Clear;
       q_Arcs.SQL.Add('Select * from "Arcfile"');
       q_Arcs.SQL.Add('where Arcfile."CLLI" = "' + CLLI + '" And Arcfile."Arc Type" = "0"');
       q_Arcs.SQL.Add('Order by Arcfile."Arc Type",Arcfile."DistToOrigin",Arcfile."Demand" DESC');
       q_Arcs.Open;

       IOF_Distance := 0;
       q_Arcs.First;
```

```
76          while Not q_Arcs.Eof do      // Loop through Node table one record a time
77            begin
78              // Add up the total distance
79              IOF_Distance := IOF_Distance + q_Arcs.FieldByName('Length').AsInteger;
80              q_Arcs.Next;
81            end;
82
83        // Setup query for fiber feeder
84          q_Arcs.Close;
85          q_Arcs.DataBaseName := 'modata';
86          q_Arcs.SQL.Clear;
87          q_Arcs.SQL.Add('Select * from "Arcfile"');
88          q_Arcs.SQL.Add('where Arcfile."CLLI" = "' + CLLI + '" And Arcfile."Arc Type" = "1"');
89          q_Arcs.SQL.Add('Order by Arcfile."Arc Type",Arcfile."DistToOrigin",Arcfile."Demand" DESC');
90          q_Arcs.Open;
91
92          Total_Distance := 0;
93          q_Arcs.First;
94          while Not q_Arcs.Eof do      // Loop through Node table one record a time
95            begin
96              // Add up the total distance
97              Total_Distance := Total_Distance + q_Arcs.FieldByName('Length').AsInteger;
98              q_Arcs.Next;
99            end;
100
101         if Total_Distance > 0
102           then begin
103                 // Calculate the length of underground, buried and aerial fiber lengths
104                 Per_Und := (IOF_Distance + Total_Distance) * DM_ICM.t_Nodes.FieldByName('F Undgnd
105 Pct').AsFloat;
106                 Per_Bur := (IOF_Distance + Total_Distance) * DM_ICM.t_Nodes.FieldByName('F Buried
107 Pct').AsFloat;
108                 Per_Aer := (IOF_Distance + Total_Distance) * DM_ICM.t_Nodes.FieldByName('F Aerial
109 Pct').AsFloat;
110
111                 // All of the inter office is buried, so we may need to adjust the placement
112                 // percentage for fiber feeder.
113                 if Per_Bur > IOF_Distance
114                   then begin  // There more buried than inter office only cable
```

```
115                         Per_Und := Per_Und / Total_Distance;
116                         Per_Aer := Per_Aer / Total_Distance;
117                       end
118                   else if (Per_Bur + Per_Aer) > IOF_Distance
119                         then begin
120                               Per_Aer := Per_Bur + Per_Aer - IOF_Distance; // Subtract inter office from
121 total buried + aerial
122                               Per_Und := Per_Und / Total_Distance;
123                               Per_Aer := Per_Aer / Total_Distance;
124                             end
125                         else begin
126                               Per_Und := Per_Bur + Per_Aer + Per_Und - IOF_Distance; // Subtract inter
127 office from total buried + aerial + undgnd
128                               Per_Und := Per_Und / Total_Distance;
129                               Per_Aer := 0.0;
130                             end;
131               end
132           else begin
133                 Per_Und := DM_ICM.t_Nodes.FieldByName('F Undgnd Pct').AsFloat;
134                 Per_Bur := DM_ICM.t_Nodes.FieldByName('F Buried Pct').AsFloat;
135                 Per_Aer := DM_ICM.t_Nodes.FieldByName('F Aerial Pct').AsFloat;
136               end;
137
138         // Length of underground placement
139         Target_Distance := Round(Total_Distance * Per_Und);
140         q_Arcs.First;
141         while (Not q_Arcs.Eof) and (Target_Distance >= MinSpan) do
142           begin    // Begin marking arcs for underground
143             Distance := q_Arcs.FieldByName('Length').AsInteger;
144             If Target_Distance >= Distance
145               then begin
146                     // The whole arc is marked for underground
147                     UpdateArc(1.0, 0.0, 0.0);
148                     Target_Distance := Target_Distance - Distance;
149                     Len := 0;
150                   end
151               else begin
152                     // Part of the arc is marked for underground
153                     Len := Distance - Target_Distance;
```

```
154                     if Len >= MinSpan
155                       then begin
156                              PA := Len / Distance;
157                              PU := 1.0 - PA;
158                            end
159                       else if len >= (MinSpan div 2)
160                              then begin
161                                     PA  := 300 / Distance;
162                                     PU  := 1.0 - PA;
163                                     Len := 300;
164                                   end
165                              else begin
166                                     PA  := 0.0;
167                                     PU  := 1.0;
168                                     Len := 0;
169                                   end;
170
171                     if (Per_Aer > 0.0)    //Check for aerial cable
172                       then UpdateArc(PU, PA, 0.0)   //Yes, Aerial cable
173                       else UpdateArc(PU, 0.0, PA);  //No, put in buried cable
174                     Target_Distance := 0;
175                   end;
176
177             q_Arcs.Next;
178           end;
179
180         // Length of aerial placement
181         Target_Distance := Round(Total_Distance * Per_Aer)-Len;
182         while (Not q_Arcs.Eof) and (Target_Distance >= MinSpan) do
183           begin    // Begin marking arcs for aerial
184             Distance := q_Arcs.FieldByName('Length').AsInteger;
185             If Target_Distance >= Distance
186               then begin
187                      // The whole arc is marked for aerial
188                      UpdateArc(0.0, 1.0, 0.0);
189                      Len := 0;
190                      Target_Distance := Target_Distance - Distance;
191                    end
192               else begin
```

```
193                     // Part of the arc is marked for aerial
194                     Len := Distance - Target_Distance;
195                     if Len >= MinSpan
196                       then begin
197                              PB := Len / Distance;
198                              PA := 1.0 - PB;
199                            end
200                       else if len >= (MinSpan div 2)
201                              then begin
202                                     PB  := 300 / Distance;
203                                     PA  := 1.0 - PB;
204                                     Len := 300;
205                                   end
206                              else begin
207                                     PB  := 0.0;
208                                     PA  := 1.0;
209                                     Len := 0;
210                                   end;
211
212                     UpdateArc(0.0, PA, PB);
213                     Target_Distance := 0;
214                   end;
215             q_Arcs.Next;
216           end;
217
218         while Not q_Arcs.Eof do
219           begin    // mark the rest of the arcs as Buried
220             UpdateArc(0.0, 0.0, 1.0);
221             q_Arcs.Next;
222           end;
223
224       //*********************************************************************
225       //                    Copper
226       //*********************************************************************
227       // Set filter for Copper
228         q_Arcs.Close;
229         q_Arcs.SQL.Clear;
230         q_Arcs.SQL.Add('Select * from "Arcfile"');
231         q_Arcs.SQL.Add('where Arcfile."CLLI" = "' + CLLI + '" And Arcfile."Arc Type" > "1"');
```

```
232         q_Arcs.SQL.Add('Order by Arcfile."Arc Type",Arcfile."DistToOrigin",Arcfile."Demand" DESC');
233         q_Arcs.Open;
234
235         Total_Distance := 0;
236         q_Arcs.First;
237         while Not q_Arcs.Eof do      // Loop through Node table one record a time
238           begin
239             // Add up the total distance
240             Total_Distance := Total_Distance + q_Arcs.FieldByName('Length').AsInteger;
241             q_Arcs.Next;
242           end;
243
244         // Length of underground placement
245         Target_Distance := Round(Total_Distance * DM_ICM.t_Nodes.FieldByName('C Undgnd Pct').AsFloat);
246         q_Arcs.First;
247         while (Not q_Arcs.Eof) and (Target_Distance >= MinSpan) do
248           begin    // Begin marking arcs for underground
249             Distance := q_Arcs.FieldByName('Length').AsInteger;
250             If Target_Distance >= Distance
251               then begin
252                      // The whole arc is marked for underground
253                      UpdateArc(1.0, 0.0, 0.0);
254                      Target_Distance := Target_Distance - Distance;
255                      Len := 0;
256                    end
257               else begin
258                      // Part of the arc is marked for underground
259                      Len := Distance - Target_Distance;
260                      if Len >= MinSpan
261                        then begin
262                               PA := Len / Distance;
263                               PU := 1.0 - PA;
264                             end
265                        else if len >= (MinSpan div 2)
266                               then begin
267                                      PA  := 300 / Distance;
268                                      PU  := 1.0 - PA;
269                                      Len := 300;
270                                    end
```

```
271                                 else begin
272                                        PA  := 0.0;
273                                        PU  := 1.0;
274                                        Len := 0;
275                                      end;
276
277                       if (Per_Aer > 0.0)     //Check for aerial cable
278                         then UpdateArc(PU, PA, 0.0)    //Yes, Aerial cable
279                         else UpdateArc(PU, 0.0, PA);  //No, put in buried cable
280                       Target_Distance := 0;
281                     end;
282
283               q_Arcs.Next;
284             end;
285
286           // Length of aerial placement
287           Target_Distance := Round(Total_Distance * DM_ICM.t_Nodes.FieldByName('C Aerial Pct').AsFloat)-Len;
288           while (Not q_Arcs.Eof) and (Target_Distance >= MinSpan) do
289             begin    // Begin marking arcs for aerial
290               Distance := q_Arcs.FieldByName('Length').AsInteger;
291               If Target_Distance >= Distance
292                 then begin
293                        // The whole arc is marked for aerial
294                        UpdateArc(0.0, 1.0, 0.0);
295                        Target_Distance := Target_Distance - Distance;
296                        Len := 0;
297                      end
298                 else begin
299                        // Part of the arc is marked for aerial
300                        Len := Distance - Target_Distance;
301                        if Len >= MinSpan
302                          then begin
303                                 PB := Len / Distance;
304                                 PA := 1.0 - PB;
305                               end
306                          else if len >= (MinSpan div 2)
307                                 then begin
308                                        PB  := 300 / Distance;
309                                        PA  := 1.0 - PB;
```

```
                                    Len := 300;
                                  end
                              else begin
                                    PB  := 0.0;
                                    PA  := 1.0;
                                    Len := 0;
                                  end;

                        UpdateArc(0.0, PA, PB);
                        Target_Distance := 0;
                      end;
              q_Arcs.Next;
            end;

         while Not q_Arcs.Eof do
            begin    // mark the rest of the arcs as Buried
              UpdateArc(0.0, 0.0, 1.0);
              q_Arcs.Next;
            end;

          DM_ICM.t_Nodes.Next;
        end;

    q_Arcs.Close;
    q_Arcs.Free;
  end;

end.
```

SKCLUSTER.pas

```
unit SKCluster;

interface
 Uses
   Classes, OSPTypes;
 Type
   TDmdPoint = record
                x        : integer;
                y        : integer;
                XinFt    : Integer;        // x-coordinate in feet of node
                YinFt    : Integer;        // y-coordinate in feet of node
                Bus_Lines: integer;
                Res_Lines: integer;
                Cluster  : Integer;
                Distance : Double;
               end;
   PDmdPoint = ^TDmdPoint;

   TKMeanPoint = record
                  x        : Double;
                  y        : Double;
                 end;

  Var
    DemandPt     : PDmdPoint;
    K_Means      : array[0..399] of TKMeanPoint;
    PrevK_Means  : array[0..399] of TKMeanPoint;
    K_Count      : array[0..399] of integer;

    LastClusterCountAdj : integer;

  function GoodResult(Data : TList; Var NumOfPoints, NumOfClusters, MaxLength :Integer) : Boolean;
  procedure K_Cluster(Data : TList; Var NumOfPoints, NumOfClusters, MaxLength :Integer);

```

```
37  implementation
38
39  //*************************************
40  function GetDistance (X1,Y1, X2,Y2 : Double) : Double;
41    begin  // Taxi Cab distance
42      Result := Abs(X1-X2) * pb.LocDist[0] + Abs(Y1-Y2) * pb.LocDist[1];
43    end;
44
45  //*************************************
46  procedure K_Cluster (Data : TList; Var NumOfPoints, NumOfClusters, MaxLength : Integer);
47    Var
48      I, K  : Integer;
49
50    //*************************************
51    function KMeans_Equal : Boolean;
52      Var
53        N, Count : Integer;
54      begin
55        Count := 0;
56        for N := 0 to NumOfClusters-1 do
57          if (K_Means[n].x = PrevK_Means[n].x) and
58             (K_Means[n].y = PrevK_Means[n].y)
59            then Inc(Count);
60        Result := Count = NumOfClusters;
61      end;
62
63    //*************************************
64    procedure Calculate_Distances;
65      Var
66        N, L     : Integer;
67        Cluster  : Integer;
68        Distance : Double;
69        Dist     : Double;
70
71      begin
72        for N := 0 to NumOfPoints-1 do
73          begin
74            DemandPt := Data.Items[N];
75            if DemandPt^.Cluster <> 0
```

```
76              then begin    // Not in core cluster
77                    Distance := 1.0E+30; // set distance to a large number
78                    Cluster  := -1;      // initial cluster index
79
80                     for L := 1 to NumOfClusters-1 do
81                       begin
82                         // Calculated distance from K_Means to demand point
83                         Dist := GetDistance(DemandPt.x, DemandPt.y, K_Means[1].x, K_Means[1].y);
84                         if Dist < Distance
85                           then begin // if this is a shorter distance, record it.
86                                  Distance := Dist;
87                                  Cluster  := L;
88                                end;
89                       end;
90
91                     DemandPt.Cluster  := Cluster;
92                     DemandPt.Distance := Distance;
93                   end;
94         end;
95     end;
96
97   //*************************************
98   procedure Calculate_Means;
99     Var
100      N : Integer;
101    begin
102      // Initialize K_Means and K_Count
103      for N := 0 to NumOfClusters-1 do
104        begin
105          K_Means[N].X := 0;
106          K_Means[N].Y := 0;
107          K_Count[N]   := 0;
108        end;
109
110      // Sum distances
111      for N := 0 to NumOfPoints-1 do
112        begin
113          DemandPt := Data.Items[N];
114          if DemandPt^.Cluster <> 0
```

```
115             then begin
116                    K_Means[DemandPt^.Cluster].X := K_Means[DemandPt^.Cluster].X + DemandPt^.x;
117                    K_Means[DemandPt^.Cluster].Y := K_Means[DemandPt^.Cluster].Y + DemandPt^.y;
118                  end;
119           Inc (K_Count[DemandPt^.Cluster]);
120         end;
121
122       // Calculated K_Means
123       for N := 1 to NumOfClusters-1 do
124         if K_Count[N] > 0
125           then begin
126                  K_Means[N].X := K_Means[N].X / K_Count[N];
127                  K_Means[N].Y := K_Means[N].Y / K_Count[N];
128                end;
129     end;
130
131   //************************************
132   procedure GetCoreCluster;
133     Var
134       N, Dist : integer;
135
136     begin
137       K_Count[0] := 0;
138
139       for N := 0 to NumOfPoints-1 do
140         begin
141           DemandPt := Data.Items[N];
142           // Calculated distance from the Wire Center Office to the demand point
143           Dist := Round(GetDistance(DemandPt.x, DemandPt.y, 0,0));
144           if Dist <= MaxLength
145             then begin // place demand point in the core.
146                    DemandPt.Distance := Dist;
147                    DemandPt.Cluster  := 0;
148                    Inc(K_Count[0]);
149                  end
150             else begin
151                    DemandPt.Distance := 0;
152                    DemandPt.Cluster  := -1;
153                  end;
```

```
154           end;
155
156       end;
157
158     //*************************************
159     Procedure SetInitialKMeans;
160       Var
161         I, N, Step  : Integer;
162
163       begin
164         for I := 0 to 399 do
165          begin
166            K_Means[I].X := 0;
167            K_Means[I].Y := 0;
168          end;
169
170         if NumOfClusters = 1 then Exit;
171         Step := NumOfPoints div NumOfClusters;
172         N := 0;
173         I := 1;
174         While  I < NumOfClusters do
175           begin
176             DemandPt      := Data.Items[N];
177             K_Means[I].X := DemandPt^.X;
178             K_Means[I].Y := DemandPt^.Y;
179             Inc(I);
180             Inc(N,Step);
181           end;
182       end;
183
184     //*************************************
185     begin   // K_Cluster
186       // Get Core Cluster
187       GetCoreCluster;
188
189       if K_Count[0] = NumOfPoints
190         then begin  // There is only 1 cluster
191                 NumOfClusters := 1;
192                 Exit;
```

```
193             end;
194
195       if LastClusterCountAdj = 0
196         then begin
197                if K_Count[0] > 0
198                  then NumOfClusters := (NumOfPoints - K_Count[0]) div K_Count[0]
199                  else NumOfClusters := 2;
200
201                If NumOfClusters < 2 then NumOfClusters := 2;
202
203                // Select initial K_Means
204                SetInitialKMeans;
205              end;
206
207       if LastClusterCountAdj < 0
208         then begin
209                If NumOfClusters < 2 then NumOfClusters := 2;
210
211                // Select initial K_Means
212                SetInitialKMeans;
213              end;
214
215        for K := 0 to NumOfClusters-1 do
216          begin
217            PrevK_Means[K].x := 0;
218            PrevK_Means[K].y := 0;
219          end;
220
221       // Start of clustering loop
222       while Not KMeans_Equal do
223         begin
224           Calculate_Distances;
225
226           // Copy current means to previous means
227           for I := 0 to NumOfClusters-1 do
228              PrevK_Means[I] := K_Means[I];
229
230           Calculate_Means;
231         end;
```

```
232     end;
233
234     //*************************************
235    function GoodResult(Data : TList; Var NumOfPoints, NumOfClusters, MaxLength :Integer) : Boolean;
236      Var
237        I, N  : Integer;
238        Dist  : Integer;
239        Good  : Integer;
240        Ratio : Double;
241
242      begin
243        if NumOfClusters = 1
244          then begin  // if there is only one cluster, it's done
245                 Result := True;
246                 Exit;
247               end;
248
249       for I := 0 to NumOfClusters-1 do
250          begin
251            PrevK_Means[I].x := 0;
252            PrevK_Means[I].y := 0;
253          end;
254
255        Good := 0;
256        for I := 0 to NumOfPoints - 1 do
257          begin
258            DemandPt := Data.Items[I];
259            Dist := round(DemandPt.Distance);
260            if Dist <= MaxLength  // check to see if in range
261              then Inc(Good)      // Count the ones in range
262              // count how many demand point are out of range for each cluster
263              else PrevK_Means[DemandPt.Cluster].x := PrevK_Means[DemandPt.Cluster].x + 1.0;
264           end;
265
266        Ratio := Good/NumOfPoints;
267         if Ratio < 0.98 // check to see if we have sufficiently converged
268              then begin  // not enough clusters
269                     N := 1;   // look for the cluster with the most outsiders
270                     for I := 2 to NumOfClusters-1 do
```

```
271                   if PrevK_Means[N].x < PrevK_Means[I].x then N := I;
272
273                 // Attempt to split the cluster
274                 I := 0;
275                 repeat
276                    DemandPt := Data.Items[I];
277                    Inc(I)
278                 Until (I = NumOfPoints) or (DemandPt.Cluster = N);
279                 K_Means[N].x :=  DemandPt.x;
280                 K_Means[N].y :=  DemandPt.y;
281
282                 repeat
283                    DemandPt := Data.Items[I];
284                    Inc(I)
285                 Until (I = NumOfPoints) or (DemandPt.Cluster = N);
286                 K_Means[NumOfClusters].x :=  DemandPt.x;
287                 K_Means[NumOfClusters].y :=  DemandPt.y;
288                 Inc(NumOfClusters);
289
290                 LastClusterCountAdj := 1;
291                 Result := False;
292                 Exit;
293               end
294           else begin  // we have a converged, but we're going to check for              RFPC0057
295                       // a cluster with too many outsiders                             RFPC0057
296                 if Ratio < 1.0
297                   then begin // if we don't have 100% convergence.                  RFPC0057
298
299                          N := 1;   // look for the cluster with the most outsiders     RFPC0057
300                          for I := 2 to NumOfClusters-1 do
301                            if PrevK_Means[N].x < PrevK_Means[I].x then N := I;
302
303                          if  (PrevK_Means[N].x / K_Count[N]) >= 0.5
304                            then begin // if 50% or more are outsiders                  RFPC0057
305                                   // Attempt to split the cluster                     RFPC0057
306                                   I := 0;                                           //RFPC0057
307                                   repeat                                            //RFPC0057
308                                      DemandPt := Data.Items[I];                     //RFPC0057
309                                      Inc(I)                                         //RFPC0057
```

```
310                                     Until (I = NumOfPoints) or (DemandPt.Cluster = N);  //RFPC0057
311                                     K_Means[N].x :=  DemandPt.x;                        //RFPC0057
312                                     K_Means[N].y :=  DemandPt.y;                        //RFPC0057
313
314                                     repeat                                              //RFPC0057
315                                        DemandPt := Data.Items[I];                       //RFPC0057
316                                        Inc(I)                                           //RFPC0057
317                                     Until (I = NumOfPoints) or (DemandPt.Cluster = N);  //RFPC0057
318                                     K_Means[NumOfClusters].x :=  DemandPt.x;            //RFPC0057
319                                     K_Means[NumOfClusters].y :=  DemandPt.y;            //RFPC0057
320                                     Inc(NumOfClusters);                                 //RFPC0057
321                                                                                         //RFPC0057
322                                     LastClusterCountAdj := 1;                           //RFPC0057
323                                     Result := False;                                    //RFPC0057
324                                     Exit;                                               //RFPC0057
325                                   end;                                                  //RFPC0057
326                           end;
327                   end;
328
329       Case LastClusterCountAdj of
330       -1,0 : begin  // too many clusters
331                 if NumOfClusters <= 2
332                   then Result := true
333                   else begin
334                          Dec(NumOfClusters);
335                          LastClusterCountAdj := -1;
336                          Result := False;
337                        end;
338               end;
339          1 : Result := True; // Just right
340       end;
341     end;
342
343   end.
344
```

SPANNING.pas

```
2
3   unit Spanning;
4
5   interface
6    Uses OSPTypes, ICMDM, Dialogs, SysUtils, Windows, OSPInvest, Invntry, Globalda, Math;
7
8       procedure Span;
9       procedure createnewnode;
10      procedure mindist(nodefrom, onaxis : Integer; Var mindistance : Integer);
11      function  IMin (I1, i2 : Integer) : Integer;
12      procedure SpanTree;
13      function  SumDemandTree(CArc : PArc) : TDemand;
14      procedure DistTree(CArc : PArc; Dist : Integer);
15      procedure DeleteTree(CArc : PArc);
16      procedure ArcTree(CArc : PArc; Point : TPoint);
17      procedure XCTree;
18      procedure ExtTree (Distance : integer);
19
20  implementation
21
22  Var
23    maxnodes      : Integer;  // number of total nodes (initial plus created)
24    tonode        : Integer;  // node number determined closest
25    newnode       : Integer;  // node number created
26    wcnode        : Integer;  // wire center or subfeeder node number (should always be 0)
27    towcnode      : Integer;  // node number closest to wire center
28    fnode         : Integer;
29    linkindex     : Integer;  // counter for link array
30
31    newx          : Integer;   // new x coordinate creating for new node
32    newy          : Integer;   // new y coordinate creating for new node
33    mintowc       : Integer;   // minimun distance to identified node from wire center
34    wc_x          : Integer;   // wire center or subfeeder x coordinate
35    wc_y          : Integer;   // wire center or subfeeder y coordinate
36    disttowc      : Integer;   // distance from node to wire center
```

```
37    disttonode    : Integer;   // distance from node to node
38    distox        : Integer;   // distance from node to x axis
39    distoy        : Integer;   // distance from node to y axis
40    disttolink    : Integer;   // distance from new node to wire center
41    closest       : Integer;   // minimum of two distance values
42    closest1      : Integer;   // minimum of two distance values
43    CurrArc       : PArc;     // Root of an Arc linked list
44    Inv           : Double;   // Cross Connect Investment
45    Found         : Boolean;
46
47  //*************************************************************************
48  //                  Constrained Spannning Tree Routine
49  //
50  //  Summary:    The Spanning Tree Routine takes input of cluster locations and
51  //              determines the minimun distance to link all of them back to the
52  //              wire center (or pair gain location for subfeeder).
53  //
54  //              The minimum distance (airline or rectilinear distance) is
55  //              constructed by finding the nearest customer node that is
56  //              not yet linked to the wire center and links it (either
57  //              directly or via a junction point).  This continues until
58  //              all customer nodes are linked back to the wire center
59  //              (directly or indirectly) by the shortest distance.
60  //
61  //              We are using the PRIM algorithm with some modifications.  The
62  //              effect of creating possible junction points (nodes) along the
63  //              North-South (Y) and East-West (X) axes emanating from the
64  //              central office (wire center).   New nodes (or junction points)
65  //              are created, as needed, along the North-South and East-West
66  //              axes and are used to link the existing nodes back to the wire
67  //              center.  The node locations (link from and link to) are saved
68  //              along with the distance between each pair of nodes.
69  //
70  //
71  //  Input File contains:    gridweight (X to Y or Long/Lat) [first record/line]
72  //                          number of clusters (or SAIs)    [second record/line]
73  //                          X, Y coordinates of each cluster (or SAI)
74  //                              (the first coordinate always being the
75  //                               wire center or subfeeder pair gain location)
```

```
76  //                              [third record to end of file]
77  //
78  //  Output File contains:    (Linked node coordinates and distance between
79  //                            one record/line per link of from, to, dist)
80  //                           X coordinate of From node
81  //                           Y coordinate of From node
82  //                           X coordinate of To node
83  //                           Y coordinate of To node
84  //                           Distance between From node and To node
85  //
86  //                             .
87  //  NOTES:
88  //  Links are made 'from' node to 'to' node where 'from' node is the one
89  //  farthest from the wire center and 'to' node is closer to the wire center.
90  //
91  //*****************************************************************************
92   function IMin (I1, i2 : Integer) : Integer;
93     begin
94       if I2 < I1
95         then Result := I2
96         else Result := I1;
97     end;
98
99  procedure Span;
100   Var
101     // Local variable definition
102     SaiTemp    : PSAI;
103     i,j        : integer;   // general loop counters
104     index      : integer;   // link array counter
105     tempx      : integer;   // temporary node created on x-axis
106     tempy      : integer;   // temporary node created on y-axis
107     tolinknode : integer;   // closest linked node to another node
108     quad1      : integer;   // flag indicating first time node located in quadrant 1
109     quad2      : integer;   // flag indicating first time node located in quadrant 2
110     quad3      : integer;   // flag indicating first time node located in quadrant 3
111     quad4      : integer;   // flag indicating first time node located in quadrant 4
112     disttempx  : Integer;    // distance from tempx to wc or closest linked node on x-axis
113     disttempy  : Integer;    // distance from tempy to wc or closest linked node on y-axis
114     distrectx  : Integer;    // rectilinear distance from node to wc or closest linked node on x-axis
```

```
115      distrecty  : Integer;    // rectilinear distance from node to wc or closest linked node on y-axis
116      dist1      : Integer;    // distance holder for min determination
117      dist2      : Integer;    // distance holder for min determination
118      dist3      : Integer;    // distance holder for min determination
119      newtowc    : Integer;    // new node number to link to wc
120      totowc     : Integer;    // node number to link to node
121      dist       : Integer;    // previous linked node distance (used when changing links)
122      TempXC     : double;    // Temporary holding place for the cross connect factor
123      nodenum    : integer;   // node number searching for in arc array creation
124      linknum    : integer;   // link (arc) number
125      arccount   : integer;   // arc counter
126      x, y, z    : word;
127      sailoop    : Integer;  // node (SAI) loop counter
128      node       : Integer;  // index counter for reading input file into SAI array
129      axisflag   : integer;  // 0=connect to x-axis; 1=connect to y-axis
130      nodeexists : integer;  // 0=new node does not exists; 1=new node exists
131      nodexexists: integer;  // 0=new node does not exists; 1=new node exists (on x axis)
132      nodeyexists: integer;  // 0=new node does not exists; 1=new node exists (on y axis)
133      Res        : integer;  // Res line count
134
135    begin
136      // initialize variables
137
138      quad1 := 0;       // no nodes in quadrant 1
139      quad2 := 0;       // no nodes in quadrant 2
140      quad3 := 0;       // no nodes in quadrant 3
141      quad4 := 0;       // no nodes in quadrant 4
142
143      SaiWCNode := SAIList.Items[0]; //Wire Center or DLC node
144      wc_x      := SaiWCNode^.x;     // save wire center X coordinate
145      wc_y      := SaiWCNode^.y;     // save wire center Y coordinate
146
147      for  node := 0 to SAI.maxSAIs-1 do
148        begin
149          SaiNode := SAIList.Items[Node];
150          SaiNode^.Node := Node;
151
152          // set indication for node being on an axis - needed for closest
153          //   linked node on axis determination
```

```
154         if (SaiNode^.x = SaiWCNode^.x) or (SaiNode^.y = SaiWCNode^.y)
155           then SaiNode^.axis := 1
156           else SaiNode^.axis := 0;
157       end;  // end for-loop
158
159     // initialize link
160     for  node := 0 to SAI.maxSAIs-1 do
161       begin
162         link[node].linkfromnode := 0;
163         link[node].fromx        := 0;
164         link[node].fromy        := 0;
165         link[node].linktonode   := 0;
166         link[node].tox          := 0;
167         link[node].toy          := 0;
168         link[node].dist         := 0;
169       end;
170
171     maxnodes  := SAI.maxSAIs;   // save off the original number of SAIs
172     linkindex := 0;             // initialize linkindex counter
173     wcnode    := 0;             // initial wire center / subfeeder node default
174
175     // find first closest node to wire center (using rectilinear distance)
176     mindist(wcnode, 0, mintowc );     // sets global variable 'tonode'
177
178     SaiNode := SAIList.Items[tonode];
179     if (SaiNode^.x >= wc_x)
180       then begin
181              if (SaiNode^.y >= wc_y)
182                then quad1 := 1
183                else if (SaiNode^.y < wc_y)
184                       then quad2 := 1;
185            end
186       else if (SaiNode^.x < wc_x)
187              then begin
188                     if (SaiNode^.y >= wc_y)
189                       then quad4 := 1
190                       else if (SaiNode^.y < wc_y)
191                              then quad3 := 1
192                  end;
```

```
193     // determine if closest node to wire center is on an axis and create
194     //  a link to the node if so
195     if (SaiNode^.x = wc_x) or (SaiNode^.y = wc_y)
196       then begin
197              SaiWcNode := SAIList.Items[wcnode];
198              link[linkindex].linkfromnode := tonode;
199              link[linkindex].fromx        := SaiNode^.x;
200              link[linkindex].fromy        := SaiNode^.y;
201              link[linkindex].linktonode   := wcnode;
202              link[linkindex].tox          := SaiWcNode.x;
203              link[linkindex].toy          := SaiWcNode.y;
204              link[linkindex].dist         := mintowc;
205              inc(linkindex);
206              SaiNode^.link   := 1;
207              SaiWcNode^.link := 1;
208            end
209       else begin
210              // closest node is not on an axis so determine which axis is closest
211              //  to node and create a new node on that axis and create links to
212              //  related nodes
213              SaiWcNode := SAIList.Items[0];
214              distox := abs( SaiNode^.XinFt - SaiWcNode^.XinFt );
215              distoy := abs( SaiNode^.YinFt - SaiWcNode^.YinFt );
216              if ( distox > distoy )
217                then begin
218                       // set values for new node on the X axis
219                       newx := SaiNode^.x;
220                       newy := wc_y;
221                       disttonode := distoy;
222                       disttowc   := distox;
223                     end
224                else begin
225                       // set values for new node on the Y axis
226                       newx := wc_x;
227                       newy := SaiNode^.y;
228                       disttonode := distox;
229                       disttowc   := distoy;
230                      end;
231
```

```
232             nodeexists := 0;
233             for node := 1 to maxnodes-1 do
234               begin
235                 SaiNode := SAIList.Items[node];
236                 if (newx = Sainode^.x)
237                   then if (newy = SaiNode^.y)
238                           then begin
239                                  nodeexists := 1;
240                                  newnode    := node;
241                                end;
242               end;
243
244             if (nodeexists = 0)  // new node does not already exist
245               then begin
246                      // create the new node
247                      createnewnode;
248                      newnode := maxnodes - 1;
249
250                      // set link from the new node to the wire center
251                      SaiNode   := SAIList.Items[newnode];
252                      SaiWCNode := SAIList.Items[wcnode];
253                      link[linkindex].linkfromnode := newnode;
254                      link[linkindex].fromx        := SaiNode^.x;
255                      link[linkindex].fromy        := SaiNode^.y;
256                      link[linkindex].linktonode   := wcnode;
257                      link[linkindex].tox          := SaiWCNode^.x;
258                      link[linkindex].toy          := SaiWCNode^.y;
259                      link[linkindex].dist         := disttowc;
260                      Inc(linkindex);
261                      SaiNode^.link                := 1;
262                    end;
263             // set link from the node identified as the closest to the
264             //   wire center to the new node
265             SaiNode   := SAIList.Items[tonode];
266             link[linkindex].linkfromnode := tonode;
267             link[linkindex].fromx        := SaiNode^.x;
268             link[linkindex].fromy        := SaiNode^.y;
269             link[linkindex].linktonode   := newnode;
270             SaiNode   := SAIList.Items[newnode];
```

```
271             link[linkindex].tox          := SaiNode^.x;
272             link[linkindex].toy          := SaiNode^.y;
273             link[linkindex].dist         := disttonode;
274             Inc(linkindex);
275
276             // set linked nodes to linked status
277             SaiNode         := SAIList.Items[tonode];
278             SaiWCNode       := SAIList.Items[wcnode];
279             SaiNode^.link   := 1;
280             SaiWCNode^.link := 1;
281           end;
282
283     // determine the next closest node to the wirecenter, create new nodes if
284     //  necessary, then create links to wirecenter via nodes
285     for sailoop := 1 to SAI.maxSAIs-1 do
286       begin
287
288
289         // find the next closest non-linked node to wire center (using rectilinear distance)
290         mindist(wcnode, 0, disttowc);  // sets global variable 'tonode'
291         towcnode := tonode;            // (next) non-linked node closest to wc
292
293         // find which linked node is closest to identified node (using airline distance)
294         mindist(towcnode, 0, disttonode);  // sets global variable 'tonode'
295         tolinknode := tonode;              // linked node closest to non-linked node
296
297         SaiNode := SAIList.Items[towcnode];
298         SaiWcNode := SAIList.Items[0];
299         distox  := abs( SaiNode^.YinFt - SaiWcNode^.YinFt );   // airline distance to x axis
300         distoy  := abs( SaiNode^.XinFt - SaiWcNode^.XinFt );   // airline distance to y axis
301
302         // create temp node on x
303         newx        := SaiNode^.x;
304         newy        := wc_y;
305         tempx       := maxnodes;
306         nodexexists := 0;
307         for node := 0 to maxnodes-1 do
308           begin
309             SaiNode := SAIList.Items[node];
```

```
310             if (newx = SaiNode^.x)
311               then if (newy = SaiNode^.y)
312                      then begin
313                             nodexexists := 1;
314                             tempx       := node;
315                           end;
316           end;
317
318         if (nodexexists = 0)     // new node does not already exist
319           then begin
320                  createnewnode;
321                  newnode := maxnodes - 1;
322                end;
323
324         // create temp node on y
325         SaiNode := SAIList.Items[towcnode];
326         newx    := wc_x;
327         newy    := SaiNode^.y;
328         tempy   := maxnodes;
329         nodeyexists := 0;
330         for node := 0 to maxnodes-1 do
331           begin
332             SaiNode := SAIList.Items[node];
333             if (newx = SaiNode^.x)
334               then if (newy = SaiNode^.y)
335                      then begin
336                             nodeyexists := 1;
337                             tempy       := node;
338                           end;
339           end;
340         if (nodeyexists = 0) and (distoy <> 0)   // new node does not already exist
341           then begin
342                  createnewnode;
343                  newnode := maxnodes - 1;
344                end;
345
346         // find which linked node is closest to tempx node (using airline distance)
347         mindist(tempx, 1, disttempx);
348         // calc X rectilinear distance to the wire center (or closest node on x-axis)
```

```
349         //   use distance to x-axis plus distance to wc/closest node on x-axis
350         distrectx := disttempx + distox;
351
352         // find which linked node is closest to tempy node (using airline distance)
353         mindist(tempy, 1, disttempy);
354         // calc Y rectilinear distance to the wire center (or closest node on y-axis)
355         //   use distance to y-axis plus distance to wc/closest node on y-axis
356         distrecty := disttempy + distoy;
357
358         // set default distance values (in case of node on an axis)
359         dist1 := disttonode;
360         dist2 := distox;
361         dist3 := distoy;
362
363         // Determine which quadrant the node is in (relative to the wire center
364         //  location).  Depending on the quadrant set the distances to use for
365         //  the minimun distance check.  If the node is the first one identified
366         //  in the quadrant use the rectilinear distance to the wire center,
367         //  distance to the x-axis, and distance to the y-axis.  Otherwise use
368         //  airline distance to the closest identified node, rectilinear distance
369         //  to the wire center (or closest linked node on the x-axis) via the
370         //  x-axis, and rectilinear distance to the wire center (or closest
371         //  linked node on the y-axis) via the y-axis.
372         SaiNode := SAIList.Items[towcnode];
373         if (SaiNode^.x >= wc_x)
374           then begin
375                  if (SaiNode^.y >= wc_y)
376                    then begin   // quadrant 1 (+,+)
377                           if (quad1 = 1)
378                             then begin
379                                    dist1 := disttonode;
380                                    dist2 := distrectx;
381                                    dist3 := distrecty;
382                                  end
383                              else begin
384                                     // first time in quadrant
385                                     dist1 := disttowc;
386                                     dist2 := distox;
387                                     dist3 := distoy;
```

```
388                                 quad1 := 1;
389                               end;
390                         end   // end quadrant 1 (+,+)
391                   else if (SaiNode^.y < wc_y)
392                     then begin // y<wc_y
393                            // quadrant 2 (+,-)
394                            if (quad2 = 1)
395                             then begin
396                                    dist1 := disttonode;
397                                    dist2 := distrectx;
398                                    dist3 := distrecty;
399                                  end
400                             else begin
401                                    // first time in quadrant
402                                    dist1 := disttowc;
403                                    dist2 := distox;
404                                    dist3 := distoy;
405                                    quad2 := 1;
406                                  end;
407                            end;    // end quadrant 2 (+,-)
408               end    // end x>wc_x
409          else if (SaiNode^.x < wc_x)
410                 then begin  // x<wc_x
411                        if (SaiNode^.y >= wc_y)
412                          then begin   // quadrant 4 (-,+)
413                                 if (quad4 = 1)
414                                   then begin
415                                          dist1 := disttonode;
416                                          dist2 := distrectx;
417                                          dist3 := distrecty;
418                                        end
419                                   else begin
420                                          // first time in quadrant
421                                          dist1 := disttowc;
422                                          dist2 := distox;
423                                          dist3 := distoy;
424                                          quad4 := 1;
425                                        end;
426                            end;    // end quadrant 4 (-,+)
```

```
427                         end
428                     else if (SaiNode^.y < wc_y)
429                           then begin // y<wc_y
430                                  // quadrant 3 (-,-)
431                                  if (quad3 = 1)
432                                    then begin
433                                           dist1 := disttonode;
434                                           dist2 := distrectx;
435                                           dist3 := distrecty;
436                                         end
437                                    else begin
438                                           // first time in quadrant
439                                           dist1 := disttowc;
440                                           dist2 := distox;
441                                           dist3 := distoy;
442                                           quad3 := 1;
443                                         end;
444                               end;   // end quadrant 3 (-,-)
445               // end x<wc_x
446
447         // determine which is closest to the non-linked node
448         closest1 := imin(dist1, dist2);
449         closest  := imin(closest1, dist3);
450
451         // if the closest is identified as a linked node and its on an axis,
452         //  use the distances to each axis (from the non-linked node) to determine
453         //  the closest axis to link to
454         SaiNode := SAIList.Items[tolinknode];
455         if (closest = dist1) and ((SaiNode^.x = wc_x) or (SaiNode^.y = wc_y))
456           then closest := imin(dist2, dist3);
457
458         // set default for axisflag depending on closest
459         if closest = dist2 //set for x axis
460           then axisflag := 0;
461         if closest = dist3 //set for y axis
462           then axisflag := 1;
463
464         // if rectilinear distances (via x & via y) are equal then check for the
465         // min distance to x axis or to y axis and set flag to connect to the
```

```
466         // min distamce
467         if (closest = distrectx) and (distrectx = distrecty)
468           then if distox > distoy    // y axis is closer
469                  then axisflag := 1
470                  else axisflag := 0;
471
472         if (closest = dist1) or (distox = 0) or (distoy = 0)
473           then begin
474                  //  if closest = the linked node OR
475                  //     non-linked node is on an axis (x or y) then link the non-linked
476                  //     node (towcnode) to linked node (tonode)
477
478                  if (nodeyexists = 0)
479                    then begin
480                           // remove the temp nodes
481                           Dec(maxnodes);
482                           SAITemp := SAIList.Items[tempy];
483                           Dispose(SAITemp);
484                           SAIList.Delete(tempy);
485                           newnode := maxnodes - 1;
486                         end;
487
488                  if (nodexexists = 0)
489                    then begin
490                           Dec(maxnodes);
491                           SAITemp := SAIList.Items[tempx];
492                           Dispose(SAITemp);
493                           SAIList.Delete(tempx);
494                           newnode := maxnodes - 1;
495                         end;
496
497                  // set link from the identified node closest to wire center (towcnode)
498                  //   to the linked node identified as closest to that node (tolinknode)
499                  SaiNode := SAIList.Items[towcnode];
500                  // set linked nodes to linked status
501                  SaiNode^.link                 := 1;
502                  link[linkindex].linkfromnode := towcnode;
503                  link[linkindex].fromx        := SaiNode^.x;
504                  link[linkindex].fromy        := SaiNode^.y;
```

```
505                SaiNode := SAIList.Items[tolinknode];
506                link[linkindex].linktonode   := tolinknode;
507                link[linkindex].tox          := SaiNode^.x;
508                link[linkindex].toy          := SaiNode^.y;
509                link[linkindex].dist         := disttonode;
510                Inc(linkindex);
511
512             end  // end if closest==disttonode
513
514          else if (closest = dist2) and (axisflag = 0)
515                then begin
516                       //  if closest = x axis then use tempx node to link (ie link to x-axis)
517                       // remove the tempy node
518                       if (nodeyexists = 0)
519                         then begin
520                                Dec(maxnodes);
521                                SAITemp := SAIList.Items[tempy];
522                                Dispose(SAITemp);
523                                SAIList.Delete(tempy);
524                                newnode := maxnodes - 1;
525                              end;
526
527                       newnode := tempx;
528                       //Check to make sure there are no nodes between newnode and towcnode
529                       SaiNode := SAIList.Items[towcnode];
530                       I := 0;
531                       while i < MaxNodes do
532                        begin
533                          SaiTemp := SAIList.Items[i];
534                          if (SaiTemp.Link = 1) and (SaiTemp.X = SaiNode.X)
535                            then begin
536                                   Dist := Abs(SaiTemp.YinFt - SaiNode.YinFt);
537                                   if Dist < distox
538                                     then begin
539                                            newnode := i;
540                                            distox  := Dist;
541                                            if (Tempx >= SAI.MaxSais) and (nodexexists = 0)
542                                              then begin
543                                                     SAITemp := SAIList.Items[tempx];
```

```
544                                         Dispose(SAITemp);
545                                         SAIList.Delete(tempx);
546                                         Dec(maxnodes);
547                                        end;
548                                end;
549                         end;
550                   Inc(I);
551                 end;
552
553               // set link from identified node closest to wire center (towcnode)
554               //   to the new node (existing or created on X axis) (newnode)
555               SaiNode := SAIList.Items[towcnode];
556               link[linkindex].linkfromnode := towcnode;
557               link[linkindex].fromx        := SaiNode^.x;
558               link[linkindex].fromy        := SaiNode^.y;
559               SaiNode := SAIList.Items[newnode];
560               link[linkindex].linktonode   := newnode;
561               link[linkindex].tox          := SaiNode^.x;
562               link[linkindex].toy          := SaiNode^.y;
563               link[linkindex].dist         := distox;
564               Inc(linkindex);
565
566               if (SaiNode^.Link = 0)
567                 then begin
568                        // identify the closest node to the new node (using airline distance)
569                        mindist(newnode, 1, disttolink);     // sets global variable 'tonode'
570
571                        // make sure the identified node is between the new node and the wc
572                        //  if it isn't make changes to the links
573                        SaiWcNode := SAIList.Items[0];
574                        SaiNode   := SAIList.Items[newnode];
575                        newtowc   := abs(SaiNode^.XinFt - SaiWcNode^.XinFt);
576                        SaiNode   := SAIList.Items[tonode];
577                        totowc    := abs(SaiNode^.XinFt - SaiWcNode^.XinFt);
578
579                        if (newtowc < totowc)
580                          then begin
581                                 for index := 0 to linkindex-2 do // loop thru existing links
582                                    begin
```

```
583                                     if (link[index].linkfromnode = tonode)   // find 'from' node
584                                       then begin
585                                              // change tonode 'from' link
586                                              tonode                  := link[index].linktonode;
587                                              dist                    := link[index].dist;
588                                              SaiNode := SAIList.Items[newnode];
589                                              link[index].linktonode := newnode;
590                                              link[index].tox         := SaiNode^.x;
591                                              link[index].toy         := SaiNode^.y;
592                                              link[index].dist        := disttolink;
593                                              disttolink              := abs(dist - disttolink);
594                                            end;  // end if
595                                         end; // end for index
596                                    end // end if newtowc<totowc
597                                 else begin
598                                        Found := False;
599                                        Index := 0;
600                                        while (Not Found) and (index < linkindex) do // loop thru existing
601  links
602                                        begin
603                                         if (link[index].linktonode = tonode)   // find 'to' node
604                                           then begin
605                                                  Fnode   := link[index].linkFromnode;
606                                                  SaiTemp := SAIList.Items[Fnode];
607                                                  SaiNode := SAIList.Items[NewNode];
608                                                  Dist    := Abs(SaiNode^.XinFt - SaiTemp^.XinFt);
609                                                  if (SaiNode^.Y = SaiTemp^.Y) and (Link[index].Dist >
610  Dist)
611                                                    then begin
612                                                           // change tonode 'to' link
613                                                           tonode                  :=
614  link[index].linktonode;
615                                                           disttolink              := Link[index].Dist -
616  Dist;
617                                                           SaiNode := SAIList.Items[newnode];
618                                                           link[index].linktonode := newnode;
619                                                           link[index].tox         := SaiNode^.x;
620                                                           link[index].toy         := SaiNode^.y;
621                                                           link[index].dist        := dist;
```

```
622                                                 Found                   := True;
623                                               end;
624                                          end;  // end if
625                                       Inc(Index);
626                                     end; // end for index
627                                  end; // end else newtowc<totowc
628
629                               // set link from the new node (newnode) to the identified linked
630                               //   (on axis) node closest to the new node (tonode)
631                               SaiNode := SAIList.Items[newnode];
632                               link[linkindex].linkfromnode := newnode;
633                               link[linkindex].fromx        := SaiNode^.x;
634                               link[linkindex].fromy        := SaiNode^.y;
635                               // set linked nodes to linked status
636                               SaiNode^.link                := 1;
637
638                               SaiNode := SAIList.Items[tonode];
639                               link[linkindex].linktonode   := tonode;
640                               link[linkindex].tox          := SaiNode^.x;
641                               link[linkindex].toy          := SaiNode^.y;
642                               link[linkindex].dist         := disttolink;
643                               Inc(linkindex);
644
645                            end;
646
647                         SaiNode := SAIList.Items[towcnode];
648                         SaiNode^.link := 1;
649                      end  // end else if closest==distox
650
651               else if (closest = dist3) and (axisflag = 1)
652                      then begin //  if closest = y axis then use tempy node to link (ie link to y-axis)
653                             newnode := tempy;
654                             if (nodexexists = 0) and (nodeyexists = 0)
655                               then begin
656                                      // move tempy node to tempx node location
657                                      SaiNode       := SAIList.Items[tempy];
658                                      SaiTemp       := SAIList.Items[tempx];
659                                      SaiTemp^      := SaiNode^;
660                                      SaiTemp^.node := SaiNode^.node - 1;
```

```
661
662                         // remove the tempy node
663                         Dec(maxnodes);
664                         SAITemp := SAIList.Items[tempy];
665                         Dispose(SAITemp);
666                         SAIList.delete(tempy);
667                         newnode := maxnodes - 1;
668                       end;
669
670                 if (nodexexists = 0) and (nodeyexists = 1)
671                   then begin
672                         // remove the tempx node of it was temporarily created
673                         Dec(maxnodes);
674                         SAITemp := SAIList.Items[tempx];
675                         Dispose(SAITemp);
676                         SAIList.delete(tempx);
677                       end;
678
679                 //Check to make sure there are no nodes between newnode and towcnode
680               SaiNode := SAIList.Items[towcnode];
681                 I := 0;
682                 While i < MaxNodes do
683                  begin
684                    SaiTemp := SAIList.Items[i];
685                    if (SaiTemp.Link = 1) and (SaiTemp.Y = SaiNode.Y)
686                      then begin
687                             Dist := Abs(SaiTemp.XinFt - SaiNode.XinFt);
688                             if Dist < distoy
689                               then begin
690                                      newnode := i;
691                                      distoy  := Dist;
692                                      if (Tempy >= SAI.MaxSais) and (nodeyexists = 0)
693                                       then begin
694                                              SAITemp := SAIList.Items[tempy];
695                                              Dispose(SAITemp);
696                                              SAIList.Delete(tempy);
697                                              Dec(maxnodes);
698                                            end;
699                                  end;
```

```
700                                     end;
701                                   Inc(I);
702                                 end;
703
704                                 // set link from identified node closest to wire center (towcnode)
705                                 //   to the new node (created on Y axis) (newnode)
706                                 SaiNode := SAIList.Items[towcnode];
707                                 link[linkindex].linkfromnode := towcnode;
708                                 link[linkindex].fromx        := SaiNode^.x;
709                                 link[linkindex].fromy        := SaiNode^.y;
710                                 SaiNode := SAIList.Items[newnode];
711                                 link[linkindex].linktonode   := newnode;
712                                 link[linkindex].tox          := SaiNode^.x;
713                                 link[linkindex].toy          := SaiNode^.y;
714                                 link[linkindex].dist         := distoy;
715                                 Inc(linkindex);
716
717                                 if (SaiNode^.Link = 0)
718                                   then begin
719                                          // identify the closest node to the new node (using airline
720   distance)
721                                          mindist(newnode, 1, disttolink);     // sets global variable
722   'tonode'
723
724                                          // make sure the identified node is between the new node and the wc
725                                          //  if it isn't make changes to the links
726                                          SaiWcNode := SAIList.Items[0];
727                                          SaiNode   := SAIList.Items[newnode];
728                                          newtowc   := abs(SaiNode^.YinFt - SaiWcNode^.YinFt);
729                                          SaiNode   := SAIList.Items[tonode];
730                                          totowc    := abs(SaiNode^.YinFt - SaiWcNode^.YinFt);
731
732                                          if (newtowc < totowc)
733                                            then begin
734                                                   for index := 0 to linkindex-2 do // loop thru existing
735   links
736                                                     begin
737                                                      if (link[index].linkfromnode = tonode)   // find 'from'
738   node
```

```
                                                    then begin
                                                           // change tonode 'from' link
                                                           tonode                  :=
link[index].linktonode;
                                                           dist                    := link[index].dist;
                                                           SaiNode := SAIList.Items[newnode];
                                                           link[index].linktonode := newnode;
                                                           link[index].tox         := SaiNode^.x;
                                                           link[index].toy         := SaiNode^.y;
                                                           link[index].dist        := disttolink;
                                                           disttolink              := abs(dist -
disttolink);
                                                         end;    // end if
                                                    end; // end for index
                                               end // end if newtowc<totowc
                                          else begin
                                                 Found := False;
                                                 Index := 0;
                                                 while (Not Found) and (index < linkindex) do // loop thru
existing links
                                                    begin
                                                     if (link[index].linktonode = tonode)   // find 'to'
node
                                                       then begin
                                                              Fnode   := link[index].linkFromnode;
                                                              SaiTemp := SAIList.Items[Fnode];
                                                              SaiNode := SAIList.Items[NewNode];
                                                              Dist    := Abs(SaiNode^.YinFt -
SaiTemp^.YinFt);
                                                              if (SaiNode^.X = SaiTemp^.X) and
(Link[index].Dist > Dist)
                                                                then begin
                                                                       // change tonode 'to' link
                                                                       tonode                :=
link[index].linktonode;
                                                                       disttolink            :=
Link[index].Dist-Dist;
                                                                       SaiNode := SAIList.Items[newnode];
                                                                       link[index].linktonode := newnode;
```

```
778                                                          link[index].tox      := SaiNode^.x;
779                                                          link[index].toy      := SaiNode^.y;
780                                                          link[index].dist     := dist;
781                                                          Found                := True;
782                                                        end;
783                                                  end;  // end if
784                                              Inc(Index);
785                                            end; // end while index
786                                          end; // end else newtowc<totowc
787
788                                        // set link from the new node (newnode) to the identified linked
789                                        //   (on axis) node closest to the new node (tonode)
790                                        SaiNode := SAIList.Items[newnode];
791                                        link[linkindex].linkfromnode := newnode;
792                                        link[linkindex].fromx        := SaiNode^.x;
793                                        link[linkindex].fromy        := SaiNode^.y;
794                                        // set linked nodes to linked status
795                                        SaiNode^.link  := 1;
796
797                                        SaiNode := SAIList.Items[tonode];
798                                        link[linkindex].linktonode   := tonode;
799                                        link[linkindex].tox          := SaiNode^.x;
800                                        link[linkindex].toy          := SaiNode^.y;
801                                        link[linkindex].dist         := disttolink;
802                                        Inc(linkindex);
803                                      end;
804
805                                  SaiNode := SAIList.Items[towcnode];
806                                  SaiNode^.link := 1;
807                                end;  // end else if closest==distoy
808        end;  // end for sailoop
809
810      // Sum up route feet for feeder
811      for i := 0 to linkindex-2 do
812        if SAI.RType <> 'F'
813           then FeedCable := FeedCable + round(link[I].dist);
814
815      SaiWCNode := SAIList.Items[0];
816
```

```
817     if Sai.maxSAIs > 1
818        then begin
819               SpanTree;
820               SumDemandTree(CurrArc);
821               DistTree(CurrArc, 0);
822
823               //Check to see if loop extenders are needed
824               if SAI.Rtype = 'S'
825                 then if Asmp.MaxLoop = 18000 //Count Loop extenders
826                        then ExtTree(16000-DM_ICM.t_Nodes.FieldByName('Distance Adj').AsInteger)
827                        else PerExtend := 0;
828
829               if (SAI.Rtype <> 'F') and ((Res_Total + Bus_Total) > Asmp.MinXSize1)
830                 then begin
831                        XCTree;
832                        if SAI.Rtype = 'S'
833                          then begin
834                                 TempXC := Asmp.XConn_Factor;
835                                 Asmp.XConn_Factor := 2.0;
836                                 Inv := XConnInvest(SAI.CLLI, SaiWCNode^.X, SaiWCNode^.Y,
837                                                    (Res_Total + Bus_Total));
838                                 Asmp.XConn_Factor := TempXC;
839
840                                 OSP_Table.Write (SAI.CLLI, '242310', 'TFFXCONN', 'F', Inv, Inv, 0.0);
841                                 OSP_Table.Write (SAI.CLLI, '242310', 'RFFXCONN', 'F',
842                                                  Inv*(Res_Total/(Res_Total + Bus_Total)),
843                                                  Inv*(Res_Total/(Res_Total + Bus_Total)), 0.0);
844                                 OSP_Table.Write (SAI.CLLI, '242310', 'BFFXCONN', 'F',
845                                                  Inv*(Bus_Total/(Res_Total + Bus_Total)),
846                                                  Inv*(Bus_Total/(Res_Total + Bus_Total)), 0.0);
847                               end;
848                      end;
849               ArcTree(CurrArc, CurrArc.Loc);
850               DeleteTree(CurrArc);
851             end;
852  end;
853 // End of the main Span routine and the beginning of the subroutines called by
854 // Span.
855
```

```
856  procedure SpanTree;
857     // Convert the Link array into a multi-threaded linked list.
858     // This will provide easy processing of the data
859   var
860    Index : integer;
861
862   procedure Run( Node, lastArc : Integer; CArc : PArc);
863     Var
864       Arc     : PArc;
865       N, I    : Integer;
866       Res     : Integer;
867       Good    : Boolean;
868
869     begin
870       Good := False;
871       N    := 0;
872       while (N < LinkIndex-1) do
873         begin
874           // Look for connecting arc
875           if Node = link[N].linktonode
876             then begin
877                    SaiNode        := SAIList.Items[N];
878                    Good           := True;
879                    New(Arc); // Create new arc
880                    // Initialize linked list pointers
881                    Arc^.Ptr1      := Nil;
882                    Arc^.Ptr2      := Nil;
883                    Arc^.Ptr3      := Nil;
884                    Arc^.Ptr4      := Nil;
885                    Arc^.Loc.X     := Round(link[N].FromX);              // Set the X coordinate
886                    Arc^.Loc.Y     := Round(link[N].FromY);              // Set the Y coordinate
887                    Arc^.Cluster.X := SaiWCNode^.X;                      // Set the X cluster
888                    Arc^.Cluster.Y := SaiWCNode^.Y;                      // Set the Y cluster
889                    Arc^.Node      := link[N].LinkFromNode;              // Set node number
890                    Arc^.Length    := round( abs(link[N].FromY - link[N].ToY) * SAI.LatDist +
891                                             abs(link[N].FromX - link[N].ToX) * SAI.LatDist *
892  SAI.gridweight);  // Set arc length
893                    SaiNode           := SAIList.Items[link[N].LinkFromNode];
894                    Arc^.Demand.Total := SaiNode^.SiteLines;
```

```
895                   Arc^.Demand.Res_Lines := Round( Arc^.Demand.Total * Res_Total / (Res_Total + Bus_Total));
896                   Arc^.Demand.Bus_Lines := Arc^.Demand.Total - Arc^.Demand.Res_Lines;
897
898                   if SAI.Rtype = 'F'
899                     then begin
900                            Arc^.Demand.Total := Ceil(Arc^.Demand.Total / MaxPg) * 4;
901                            Arc^.OSPType      := 'F';
902                          end
903                     else Arc^.OSPType := 'D';
904
905                   //find free linked list pointer
906                   if CArc^.Ptr1 = Nil
907                     then CArc^.Ptr1 := Arc
908                     else if CArc^.Ptr2 = Nil
909                            then CArc^.Ptr2 := Arc
910                            else if CArc^.Ptr3 = Nil
911                                   then CArc^.Ptr3 := Arc
912                                   else if CArc^.Ptr4 = Nil
913                                          then CArc^.Ptr4 := Arc
914                                          else MessageDlg('Ran out of links', mtError, [mbOk], 0);
915                   run(link[N].linkfromnode, N, Arc) //update new arc
916                 end;
917           Inc(N);
918         end;
919       if Not Good
920         then begin
921                SaiNode := SAIList.Items[node];
922                CArc^.Loc.X        := SaiNode^.X; // Set the X coordinate
923                CArc^.Loc.Y        := SaiNode^.Y; // Set the Y coordinate
924                CArc^.Cluster.X    := SaiWCNode^.X;       // Set the X cluster
925                CArc^.Cluster.Y    := SaiWCNode^.Y;       // Set the Y cluster
926                CArc^.Node         := Node;               // Set node number
927                CArc^.Length       := round( abs(link[LastArc].FromY-link[LastArc].ToY)*SAI.LatDist+
928                                             abs(link[LastArc].FromX-
929   link[LastArc].ToX)*SAI.LatDist*SAI.gridweight);  // Set arc length
930                CArc^.Demand.Total := SaiNode^.SiteLines;
931                CArc^.Demand.Res_Lines := Round( Arc^.Demand.Total * Res_Total / (Res_Total + Bus_Total));
932                CArc^.Demand.Bus_Lines := Arc^.Demand.Total - Arc^.Demand.Res_Lines;
933
```

```
934                 if SAI.Rtype = 'F'
935                   then begin
936                          Arc^.Demand.Total := Ceil(Arc^.Demand.Total / MaxPg) * 4;
937                          Arc^.OSPType      := 'F';
938                        end
939                   else Arc^.OSPType := 'D';
940               end;
941     end;
942
943   begin
944     New(CurrArc);  // Create the root of a linked list
945     // Initialize linked list pointers
946     CurrArc^.Ptr1      := Nil;
947     CurrArc^.Ptr2      := Nil;
948     CurrArc^.Ptr3      := Nil;
949     CurrArc^.Ptr4      := Nil;
950     CurrArc^.Loc.X     := SaiWCNode^.X;       // Set the X coordinate
951     CurrArc^.Loc.Y     := SaiWCNode^.Y;       // Set the Y coordinate
952     CurrArc^.Cluster.X := SaiWCNode^.X;       // Set the X cluster
953     CurrArc^.Cluster.Y := SaiWCNode^.Y;       // Set the Y cluster
954     CurrArc^.Node      := 0;                  // Set node number
955     CurrArc^.Length    := 0;                  // Set arc length
956     CurrArc^.Demand.Total := 0;
957     CurrArc^.Demand.Res_Lines := 0;
958     CurrArc^.Demand.Bus_Lines := 0;
959     run(0,0, CurrArc);
960   end;  // end of SpanTree
961
962 procedure DeleteTree(CArc : PArc);
963     // Remove the  multi-threaded linked list.
964   begin
965     if CArc^.Ptr1 <> Nil then DeleteTree (CArc^.Ptr1); //Remove all arcs off of this branch
966     if CArc^.Ptr2 <> Nil then DeleteTree (CArc^.Ptr2); //Remove all arcs off of this branch
967     if CArc^.Ptr3 <> Nil then DeleteTree (CArc^.Ptr3); //Remove all arcs off of this branch
968     if CArc^.Ptr4 <> Nil then DeleteTree (CArc^.Ptr4); //Remove all arcs off of this branch
969     Dispose(CArc);
970   end;  // end of Deletetree
971
972 procedure DistTree(CArc : PArc; Dist : Integer);
```

```
973     // Calculate distance from origin.
974   begin
975     CArc^.DistToOrigin := Dist;
976     Dist := Dist + CArc^.Length;
977     if CArc^.Ptr1 <> Nil then DistTree (CArc^.Ptr1, Dist); //Calculate distance off of this branch
978     if CArc^.Ptr2 <> Nil then DistTree (CArc^.Ptr2, Dist); //Calculate distance off of this branch
979     if CArc^.Ptr3 <> Nil then DistTree (CArc^.Ptr3, Dist); //Calculate distance off of this branch
980     if CArc^.Ptr4 <> Nil then DistTree (CArc^.Ptr4, Dist); //Calculate distance off of this branch
981   end;  // end of Disttree
982
983 function SumDemandTree(CArc : PArc) : TDemand;
984   Var
985     Demand : TDemand;
986
987   begin
988     CArc^.AccumDemand.Total     := 0;
989     CArc^.AccumDemand.Res_Lines := 0;
990     CArc^.AccumDemand.Bus_Lines := 0;
991
992     if CArc^.Ptr1 <> Nil
993       then begin
994              Demand := SumDemandTree (CArc^.Ptr1); //Sum demand off of this branch
995              CArc^.AccumDemand.Total     := CArc^.AccumDemand.Total     + Demand.Total;
996              CArc^.AccumDemand.Res_Lines := CArc^.AccumDemand.Res_Lines + Demand.Res_Lines;
997              CArc^.AccumDemand.Bus_Lines := CArc^.AccumDemand.Bus_Lines + Demand.Bus_Lines;
998            end;
999
1000    if CArc^.Ptr2 <> Nil
1001      then begin
1002             Demand := SumDemandTree (CArc^.Ptr2); //Sum demand off of this branch
1003             CArc^.AccumDemand.Total     := CArc^.AccumDemand.Total     + Demand.Total;
1004             CArc^.AccumDemand.Res_Lines := CArc^.AccumDemand.Res_Lines + Demand.Res_Lines;
1005             CArc^.AccumDemand.Bus_Lines := CArc^.AccumDemand.Bus_Lines + Demand.Bus_Lines;
1006           end;
1007
1008    if CArc^.Ptr3 <> Nil
1009      then begin
1010             Demand := SumDemandTree (CArc^.Ptr3); //Sum demand off of this branch
1011             CArc^.AccumDemand.Total     := CArc^.AccumDemand.Total     + Demand.Total;
```

```
1012              CArc^.AccumDemand.Res_Lines := CArc^.AccumDemand.Res_Lines + Demand.Res_Lines;
1013              CArc^.AccumDemand.Bus_Lines := CArc^.AccumDemand.Bus_Lines + Demand.Bus_Lines;
1014            end;
1015
1016      if CArc^.Ptr4 <> Nil
1017        then begin
1018              Demand := SumDemandTree (CArc^.Ptr4); //Sum demand off of this branch
1019              CArc^.AccumDemand.Total     := CArc^.AccumDemand.Total     + Demand.Total;
1020              CArc^.AccumDemand.Res_Lines := CArc^.AccumDemand.Res_Lines + Demand.Res_Lines;
1021              CArc^.AccumDemand.Bus_Lines := CArc^.AccumDemand.Bus_Lines + Demand.Bus_Lines;
1022            end;
1023
1024      CArc^.AccumDemand.Total     := CArc^.AccumDemand.Total     + CArc^.Demand.Total;
1025      CArc^.AccumDemand.Res_Lines := CArc^.AccumDemand.Res_Lines + CArc^.Demand.Res_Lines;
1026      CArc^.AccumDemand.Bus_Lines := CArc^.AccumDemand.Bus_Lines + CArc^.Demand.Bus_Lines;
1027      SumDemandTree := CArc^.AccumDemand;
1028    end;  // end of SumDemandTree
1029
1030  procedure ArcTree(CArc : PArc; Point : TPoint);
1031      // Output arcs
1032    begin
1033      if (CArc^.Loc.X <> Point.X) or (CArc^.Loc.Y <> Point.Y)
1034        then begin
1035              DM_ICM.t_Arcs.Append;
1036              DM_ICM.t_Arcs.FieldByName('CLLI').AsString       := SAI.Clli;
1037              Case SAI.RType of
1038               'F' : DM_ICM.t_Arcs.FieldByName('Arc Type').AsInteger := 1;
1039               'W' : DM_ICM.t_Arcs.FieldByName('Arc Type').AsInteger := 2;
1040               'S' : DM_ICM.t_Arcs.FieldByName('Arc Type').AsInteger := 3;
1041              end;
1042              DM_ICM.t_Arcs.FieldByName('OSP Type').AsString   := CArc^.OSPType;
1043              DM_ICM.t_Arcs.FieldByName('X').AsInteger         := CArc^.Cluster.X;
1044              DM_ICM.t_Arcs.FieldByName('Y').AsInteger         := CArc^.Cluster.Y;
1045              DM_ICM.t_Arcs.FieldByName('From X').AsInteger    := Point.X;
1046              DM_ICM.t_Arcs.FieldByName('From Y').AsInteger    := Point.Y;
1047              DM_ICM.t_Arcs.FieldByName('To X').AsInteger      := CArc^.Loc.X;
1048              DM_ICM.t_Arcs.FieldByName('To Y').AsInteger      := CArc^.Loc.Y;
1049              DM_ICM.t_Arcs.FieldByName('Node').AsInteger      := CArc^.Node;
1050              DM_ICM.t_Arcs.FieldByName('Length').AsInteger    := CArc^.Length;
```

```
1051              DM_ICM.t_Arcs.FieldByName('LoopLenCode').AsInteger  := CArc^.Length *
1052                                          (CArc^.AccumDemand.Res_Lines + CArc^.AccumDemand.Bus_Lines);
1053              DM_ICM.t_Arcs.FieldByName('DistToOrigin').AsInteger := CArc^.DistToOrigin;
1054              DM_ICM.t_Arcs.FieldByName('Demand').AsInteger    := CArc^.AccumDemand.Total;
1055              DM_ICM.t_Arcs.FieldByName('Res Lines').AsInteger := CArc^.AccumDemand.Res_Lines;
1056              DM_ICM.t_Arcs.FieldByName('Bus Lines').AsInteger := CArc^.AccumDemand.Bus_Lines;
1057              DM_ICM.t_Arcs.FieldByName('BedRock').AsInteger   := BedRock;
1058              DM_ICM.t_Arcs.FieldByName('WaterTable').AsInteger:= WaterTable;
1059              DM_ICM.t_Arcs.FieldByName('Elevation').AsInteger := Elevation;
1060              DM_ICM.t_Arcs.FieldByName('Highway').AsInteger   := Highway;
1061              if Asmp.HML = 'L'
1062               then DM_ICM.t_Arcs.FieldByName('CanPlow').AsInteger   := 1
1063               else DM_ICM.t_Arcs.FieldByName('CanPlow').AsInteger   := 0;
1064              DM_ICM.t_Arcs.FieldByName('Structure').AsInteger := Asmp.Structure_Flag;
1065              DM_ICM.t_Arcs.FieldByName('IOF').AsInteger       := 0;
1066              DM_ICM.t_Arcs.FieldByName('HRF').AsInteger       := 0;
1067              DM_ICM.t_Arcs.FieldByName('IRF').AsInteger       := 0;  // RFPC0067
1068              DM_ICM.t_Arcs.Post;
1069            end;
1070
1071     if CArc^.Ptr1 <> Nil then ArcTree (CArc^.Ptr1, CArc^.Loc); //Output arcs off of this branch
1072     if CArc^.Ptr2 <> Nil then ArcTree (CArc^.Ptr2, CArc^.Loc); //Output arcs off of this branch
1073     if CArc^.Ptr3 <> Nil then ArcTree (CArc^.Ptr3, CArc^.Loc); //Output arcs off of this branch
1074     if CArc^.Ptr4 <> Nil then ArcTree (CArc^.Ptr4, CArc^.Loc); //Output arcs off of this branch
1075   end;  // end of Arctree
1076
1077 procedure XCTree;
1078   Var
1079     RDemand  : Integer;
1080     LastXC   : Boolean;
1081     XCLabel  : String;
1082
1083   function Run(CArc : PArc) : Char;
1084    Var
1085      FD : Char;
1086      // Check for Cross Connect Box placements.
1087    begin
1088      FD := 'D';
1089      if (CArc^.Ptr1 = Nil) and
```

```
1090          (CArc^.Ptr2 = Nil) and
1091          (CArc^.Ptr3 = Nil) and
1092          (CArc^.Ptr4 = Nil) and
1093          ((CArc^.DistToOrigin + CArc^.Length) > Asmp.XCMinDist)
1094        then LastXC  := False
1095        else LastXC  := True;
1096
1097      if CArc^.Ptr1 <> Nil then if Run (CArc^.Ptr1) = 'F' then FD := 'F';
1098      if CArc^.Ptr2 <> Nil then if Run (CArc^.Ptr2) = 'F' then FD := 'F';
1099      if CArc^.Ptr3 <> Nil then if Run (CArc^.Ptr3) = 'F' then FD := 'F';
1100      if CArc^.Ptr4 <> Nil then if Run (CArc^.Ptr4) = 'F' then FD := 'F';
1101
1102      if FD = 'D'
1103        then begin
1104               RDemand := 0;
1105               if (CArc^.DistToOrigin + CArc^.Length) > Asmp.XCMinDist
1106                 then LastXC  := False;
1107             end;
1108
1109      if (((CArc^.AccumDemand.Total-RDemand) * 3) > MaxXC) and
1110         ((CArc^.DistToOrigin+CArc^.Length) > Asmp.XCMinDist)
1111        then begin
1112               FD  := 'F';
1113               Inv := XConnInvest(SAI.CLLI, CArc^.Cluster.X, CArc^.Cluster.Y,
1114                                  CArc^.AccumDemand.Total - RDemand -
1115                                  CArc^.Demand.Total);
1116               RDemand := CArc^.AccumDemand.Total - CArc^.Demand.Total;
1117
1118               OSP_Table.Write (SAI.CLLI, '242310', 'TF'+XCLabel, 'F', Inv, Inv, 0.0);
1119               OSP_Table.Write (SAI.CLLI, '242310', 'RF'+XCLabel, 'F',
1120                                Inv*(CArc^.AccumDemand.Res_Lines/CArc^.AccumDemand.Total),
1121                                Inv*(CArc^.AccumDemand.Res_Lines/CArc^.AccumDemand.Total), 0.0);
1122               OSP_Table.Write (SAI.CLLI, '242310', 'BF'+XCLabel, 'F',
1123                                Inv*(CArc^.AccumDemand.Bus_Lines/CArc^.AccumDemand.Total),
1124                                Inv*(CArc^.AccumDemand.Bus_Lines/CArc^.AccumDemand.Total), 0.0);
1125             end
1126
1127        else begin
1128               if ((CArc^.DistToOrigin+CArc^.Length) <= Asmp.XCMinDist)and (Not LastXC)
```

```
1129                    then begin
1130                           LastXC  := True;
1131                           if (CArc^.AccumDemand.Total - RDemand) >  Asmp.MinXSize2
1132                             then begin
1133                                    FD      := 'F';
1134                                    Inv     := XConnInvest(SAI.CLLI, CArc^.Cluster.X, CArc^.Cluster.Y,
1135                                                           CArc^.AccumDemand.Total - RDemand);
1136                                    RDemand := CArc^.AccumDemand.Total - CArc^.Demand.Total;
1137
1138                                    OSP_Table.Write (SAI.CLLI, '242310', 'TF'+XCLabel, 'F', Inv, Inv, 0.0);
1139                                    OSP_Table.Write (SAI.CLLI, '242310', 'RF'+XCLabel, 'F',
1140
1141  Inv*(CArc^.AccumDemand.Res_Lines/CArc^.AccumDemand.Total),
1142
1143  Inv*(CArc^.AccumDemand.Res_Lines/CArc^.AccumDemand.Total), 0.0);
1144                                    OSP_Table.Write (SAI.CLLI, '242310', 'BF'+XCLabel, 'F',
1145
1146  Inv*(CArc^.AccumDemand.Bus_Lines/CArc^.AccumDemand.Total),
1147
1148  Inv*(CArc^.AccumDemand.Bus_Lines/CArc^.AccumDemand.Total), 0.0);
1149                                  end;
1150                          end;
1151                 end;
1152
1153        CArc^.OSPType := FD;
1154        Result := FD;
1155     end;  // end of Run
1156
1157   begin
1158     if SAI.Rtype = 'S'
1159       then XCLabel := 'FXCONN'
1160       else XCLabel := 'CXCONN';
1161
1162     RDemand := 0; LastXC := False;
1163     if CurrArc^.Ptr1 <> Nil then Run (CurrArc^.Ptr1);
1164
1165     RDemand := 0; LastXC := False;
1166     if CurrArc^.Ptr2 <> Nil then Run (CurrArc^.Ptr2);
1167
```

```
1168      RDemand := 0; LastXC := False;
1169      if CurrArc^.Ptr3 <> Nil then Run (CurrArc^.Ptr3);
1170
1171      RDemand := 0; LastXC := False;
1172      if CurrArc^.Ptr4 <> Nil then Run (CurrArc^.Ptr4);
1173    end;  // end of XCTree
1174
1175    procedure ExtTree (Distance : integer);
1176     procedure Run(CArc : PArc);
1177        // Check for Loop extenders.
1178      begin
1179        if (CArc^.DistToOrigin+CArc^.Length) > Distance
1180          then PerExtend := PerExtend + CArc^.AccumDemand.Total
1181          else begin
1182                 if CArc^.Ptr1 <> Nil then Run (CArc^.Ptr1);
1183                 if CArc^.Ptr2 <> Nil then Run (CArc^.Ptr2);
1184                 if CArc^.Ptr3 <> Nil then Run (CArc^.Ptr3);
1185                 if CArc^.Ptr4 <> Nil then Run (CArc^.Ptr4);
1186               end;
1187      end;  // end of Run
1188    begin
1189      PerExtend := 0;
1190      if CurrArc^.Ptr1 <> Nil then Run (CurrArc^.Ptr1);
1191      if CurrArc^.Ptr2 <> Nil then Run (CurrArc^.Ptr2);
1192      if CurrArc^.Ptr3 <> Nil then Run (CurrArc^.Ptr3);
1193      if CurrArc^.Ptr4 <> Nil then Run (CurrArc^.Ptr4);
1194    end;  // end of ExtTree
1195
1196  //-------------------------------------------------------------------------
1197  //-------------------------------------------------------------------------
1198
1199
1200  procedure mindist(nodefrom, onaxis : Integer; Var mindistance : Integer);
1201  //*************************************************************************/
1202  // SUMMARY:
1203  //      Find the closest node to the passed node.  This will check the complete
1204  //      SAI node array (even new nodes created if 'maxnodes' is current node
1205  //      count).  When the wire center is sent as the 'nodefrom' only original,
1206  //      non-linked SAI nodes are checked and the rectilinear distance from
```

```
1207  //      the node back to the wire center is tested.  If a node other than the
1208  //      wire center node is sent as the 'nodefrom' then only linked nodes
1209  //      (original and new), using the airline distance, will be checked.
1210  //      'tonode' is a global value and is set in this function
1211  // PARAMETERS:
1212  //    (passed to mindist)
1213  //      nodefrom        node number from which to find closest
1214  //      onaxis          0 = check all nodes
1215  //                      1 = check only nodes identified as on an axis
1216  //    (returned from mindist)
1217  //      mindistance     rectilinear or airline distance between 'nodefrom'
1218  //                       and 'tonode'
1219  //*************************************************************************/
1220
1221   Var
1222   nodeloop   : integer;              // node loop counter
1223   Distance   : integer;
1224   X1, X2     : double;
1225   Y1, Y2     : double;
1226   SAI1, SAI2 : PSAI;
1227
1228   begin
1229     mindistance := $7FFFFFFF;  // largest integer
1230     tonode := 0;
1231     SAI1 := SAIList.Items[nodefrom];
1232
1233     for nodeloop := 0 to maxnodes-1 do
1234       begin
1235         if (nodefrom <> nodeloop)   // don't check it's self
1236           then begin
1237                  SAI2 := SAIList.Items[nodeloop];
1238                  if (nodefrom = wcnode) and (SAI2^.link = 0)
1239                    then begin
1240                           //  if from wc and to node not-linked (check rectilinear distance)
1241                           Distance := abs(SAI1^.XinFt - SAI2^.XinFt) + abs(SAI1^.YinFt - SAI2^.YinFt);
1242                           if (Distance < mindistance)
1243                              then begin
1244                                     mindistance := Distance;
1245                                     tonode := nodeloop;
```

```
1246                                    end;
1247                            end;  // end from wc node and not linked
1248
1249                    if (nodefrom <> wcnode) and (SAI2^.link = 1)
1250                      then begin
1251                             //  if from not wc and to node is linked (check airline distance)
1252                             X1 := SAI1^.XinFt; X2 := SAI2^.XinFt;
1253                             Y1 := SAI1^.YinFt; Y2 := SAI2^.YinFt;
1254                             Distance := round(Sqrt(sqr(X1 - X2) + sqr(Y1 - Y2)));
1255                             if (onaxis = 0)
1256                               then begin  // check all linked nodes
1257                                      if (Distance < mindistance)
1258                                        then begin
1259                                               mindistance := Distance;
1260                                               tonode := nodeloop;
1261                                             end;
1262                                    end  // end onaxis==0
1263                               else if (onaxis = 1)
1264                                      then begin  // check only linked nodes on an axis
1265                                             if SAI2^.axis = 1
1266                                               then begin  // node on an axis
1267                                                      if (Distance < mindistance)
1268                                                        then begin
1269                                                               mindistance := Distance;
1270                                                               tonode := nodeloop;
1271                                                             end;
1272                                                  end;   // end sai.axis==1
1273                                           end;  // end else if onaxis==1
1274                            end     // end link==1
1275                    end;  // end nodefrom <> nodeloop
1276          end;  // next nodeloop
1277      end;   // end mindist function
1278
1279  //-------------------------------------------------------------------------
1280  //-------------------------------------------------------------------------
1281
1282
1283
1284  procedure createnewnode;
```

```
1285  //*****************************************************************************/
1286  // SUMMARY:
1287  //      Creates a new node (usually on an axis) using the newx and newy
1288  //      global values as the node location and save this node in the
1289  //      SAI array.
1290  // PARAMETERS:  None  (all variables are global)
1291  //*****************************************************************************/
1292   begin
1293     New(SaiNode);
1294     SaiNode^.Node  := maxnodes;
1295     SaiNode^.X     := newx;
1296     SaiNode^.Y     := newy;
1297     SaiNode^.XinFt := newx * round(Pb.LocDist[0]);
1298     SaiNode^.YinFt := newy * round(Pb.LocDist[1]);
1299     SaiNode^.Link  := 0;     // new node not linked yet
1300     SaiNode^.Axis  := 1;     // new node always on axis
1301     SaiNode^.SiteLines := 0;
1302     SAIList.Add(SaiNode);
1303     Inc(maxnodes);
1304   end;   // end createnewnode function
1305
1306  end.
1307
```

SS7.pas

```
unit Ss7;
{
RFPC0018
Description of the program change desired:

  · SS7 investment and utilization data that is currently
    entered state-by-state through input spreadsheet files
    will be calculated within the module, eliminating need
    for state-specific pre-processing in a spreadsheet to
    produce SS7 input files.

  · Attached equations would replace all current SS7-related
    calculations, except for the application of capital cost
    and expense factors and the conversion of busy hour costs
    to rate period-specific costs.

How will the desired change enhance the Integrated Cost Model (ICM):

  · Accuracy of SS7 network element and service costs will be optimized.

  · Signaling Transfer Point (STP) costs will be vendor-specific
    and will be disaggregated based upon specific cost drivers.

  · The multiple state SS7 network can be studied efficiently for
    different states.

  · Location-specific costs will be available for STPs and databases
    located outside the state being studied; wherever possible, all
    STP-specific costs will be constant values and will no longer
    vary with the office being studied.

  · Average SS7 costs will reflect actual measured utilization of
    SS7 links, as opposed to estimated utilizations.
```

```
37      · Costs for SS7 link routes will be forward-looking costs based
38        on current monthly lease amounts.  The formula now in the model
39        is based on historic costs and a current cost to historic cost
40        ratio to estimate forward-looking costs.
41
42    |------------------------------------
43    RFPC0060
44        1.-The ICM Switch Module is changed to process site specific switch
45           investment. Site specific model investment data replaces the
46           current generic switch models. Related changes in SS7 module are
47           as follows:
48          *Removed Switch Type substitution. Switch Type substitution is no
49           longer allowed in ICM, except in Costmod and SCIS.
50          *T_Sw_EFI table is no longer needed in this module. EFI factor is
51           read from the Switch Investment table t_swcost. EFI is under a new
52           investment type M0004.
53    }
54
55    interface
56
57    uses
58      SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
59      Forms, Dialogs, StdCtrls, ExtCtrls, DBCtrls, Math, DBGrids, Menus, DB,
60      DBTables, Printers, Assoc, GenPrnt, Gauges, Query, Gen_Exp, Import, Monitr,
61      Tabs, ICMDM, Globalda, Grids;
62
63    type
64      TInvestment = array[1..3] of Double;
65
66      Tf_ss7 = class(TForm)
67        l_CopyRight: TLabel;
68        ds_SS7: TDataSource;
69        sd_SS7: TSaveDialog;
70        od_SS7: TOpenDialog;
71        MainMenu1: TMainMenu;
72        File1: TMenuItem;
73        New1: TMenuItem;
74        Open1: TMenuItem;
75        N1: TMenuItem;
```

```
76      Import1: TMenuItem;
77      Export1: TMenuItem;
78      N2: TMenuItem;
79      Exit1: TMenuItem;
80      Help1: TMenuItem;
81      dbg_SS7: TDBGrid;
82      dbn_SS7: TDBNavigator;
83      cb_Assoc: TCheckBox;
84      Print1: TMenuItem;
85      N3: TMenuItem;
86      PrintDialog1: TPrintDialog;
87      p_Status: TPanel;
88      Gauge1: TGauge;
89      Query1: TMenuItem;
90      ts_SS7: TTabSet;
91      t_T1_SS7: TTable;
92      Common1: TMenuItem;
93      Tab1: TMenuItem;
94      procedure FormResize(Sender: TObject);
95      procedure FormShow(Sender: TObject);
96      procedure Exit1Click(Sender: TObject);
97      procedure FormHide(Sender: TObject);
98      procedure New1Click(Sender: TObject);
99      procedure Open1Click(Sender: TObject);
100     procedure Import1Click(Sender: TObject);
101     procedure Export1Click(Sender: TObject);
102     procedure Print1Click(Sender: TObject);
103     procedure Query1Click(Sender: TObject);
104     procedure FormActivate(Sender: TObject);
105     procedure ts_SS7Change(Sender: TObject; NewTab: Integer;
106       var AllowChange: Boolean);
107     procedure Common1Click(Sender: TObject);
108     procedure Tab1Click(Sender: TObject);
109   private
110     { Private declarations }
111     procedure SetCaption(TableIndex : Integer);
112     procedure Make_SS7(TableIndex : Integer);
113     procedure CreateActSS7;
114     procedure CreateTmpSS7;
```

```
115       procedure SS7_Invst_Nodes;
116       procedure SS7_Invst_STP;
117       procedure SS7_Invst_SCP;
118       procedure OutputHostInvestments;
119       procedure OutputRemoteInvestments;
120       procedure WriteTemp(CLLI_Code, Item: String; Inv : TInvestment);
121       procedure WriteSS7Act(t_Table : TTable; CLLI_Code, Item, Account: String; Inv : TInvestment);
122       procedure LoadH2HValues;
123       procedure GetInterOfficeValues;
124       procedure OpenFiles(Proc : Boolean);
125       function  GetParameter(S : String) : Double;
126       function  GetSS7Temp(CLLI_Code, S : String) : Boolean;
127       function  GetDistance(CLLI1, CLLI2 : String) : Double;
128    public
129       { Public declarations }
130       procedure Entry;
131       Procedure Process;
132       procedure PrintSS7( Open : Boolean);
133     end;
134
135  var
136    f_ss7: Tf_ss7;
137    TabName : String;
138    Rec_Cnt  : LongInt;
139
140  implementation
141  uses nemain;
142  {$R *.DFM}
143  Var
144
145   //RFPC0018 Inter - Office data at a state level
146    SS7_St_IHCBDS0    : TInvestment; //RFPC0018 Number of Channel Bank DS-0
147    SS7_St_IHDSX1DS1  : TInvestment; //RFPC0018 Number of DS-1 Manual Cross Connect DSX - 1
148    SS7_St_IHDSX3DS3  : TInvestment; //RFPC0018 Number of DS-3 Manual Cross Connect DSX - 3
149    SS7_St_IHFDP      : TInvestment; //RFPC0018 Fiber Distribution Panel
150    SS7_St_IHADM      : TInvestment; //RFPC0018 SONET Add/Drop Multiplexer
151    SS7_St_IHDSX3     : TInvestment; //RFPC0018 Manual Cross Connect DSX - 3
152    SS7_St_IHDCS31    : TInvestment; //RFPC0018 Digital Cross Connect System DCS3/1
153    SS7_St_IHMUX31    : TInvestment; //RFPC0018 3/1 Multiplexer MUX3/1
```

```
154     SS7_St_IHDSX1      : TInvestment; //RFPC0018 Manual Cross Connect DSX - 1
155     SS7_St_IHDCS10     : TInvestment; //RFPC0018 Digital Cross Connect System DCS1/0
156     SS7_St_IHCB        : TInvestment; //RFPC0018 Channel Bank
157     SS7_St_IHAERFI     : TInvestment; //RFPC0018 Aerial fiber
158     SS7_St_IHBURFI     : TInvestment; //RFPC0018 Buried fiber
159     SS7_St_IHUNDFI     : TInvestment; //RFPC0018 Underground fiber
160     SS7_St_IHAERDIST   : TInvestment; //RFPC0018 Length of Aerial fiber
161     SS7_St_IHBURDIST   : TInvestment; //RFPC0018 Length of Buried fiber
162     SS7_St_IHUNGDIST   : TInvestment; //RFPC0018 Length of Underground fiber
163     SS7_St_IHPOLES     : TInvestment; //RFPC0018 Poles
164     SS7_St_IHCOND      : TInvestment; //RFPC0018 Conduit
165     SS7_St_IHDS0       : TInvestment; //RFPC0018 DS0
166     SS7_St_IHDS1       : TInvestment; //RFPC0018 DS1
167     SS7_St_IHDS3       : TInvestment; //RFPC0018 DS3
168     SS7_St_IHSWPORTS   : TInvestment; //RFPC0018 switch ports
169     SS7_St_DSODistance: TInvestment; //RFPC0018 DS0 weighted distance
170
171    //RFPC0018 Inter - Office data at a state or node level
172     IHCBDS0            : TInvestment; //RFPC0018 Number of Channel Bank DS-0
173     IHDSX1DS1          : TInvestment; //RFPC0018 Number of DS-1 Manual Cross Connect DSX - 1
174     IHDSX3DS3          : TInvestment; //RFPC0018 Number of DS-3 Manual Cross Connect DSX - 3
175     IHFDP              : TInvestment; //RFPC0018 Fiber Distribution Panel
176     IHADM              : TInvestment; //RFPC0018 SONET Add/Drop Multiplexer
177     IHDSX3             : TInvestment; //RFPC0018 Manual Cross Connect DSX - 3
178     IHDCS31            : TInvestment; //RFPC0018 Digital Cross Connect System DCS3/1
179     IHMUX31            : TInvestment; //RFPC0018 3/1 Multiplexer MUX3/1
180     IHDSX1             : TInvestment; //RFPC0018 Manual Cross Connect DSX - 1
181     IHDCS10            : TInvestment; //RFPC0018 Digital Cross Connect System DCS1/0
182     IHCB               : TInvestment; //RFPC0018 Channel Bank
183     IHAERFI            : TInvestment; //RFPC0018 Aerial fiber
184     IHBURFI            : TInvestment; //RFPC0018 Buried fiber
185     IHUNDFI            : TInvestment; //RFPC0018 Underground fiber
186     IHAERDIST          : TInvestment; //RFPC0018 Length of Aerial fiber
187     IHBURDIST          : TInvestment; //RFPC0018 Length of Buried fiber
188     IHUNGDIST          : TInvestment; //RFPC0018 Length of Underground fiber
189     IHPOLES            : TInvestment; //RFPC0018 Poles
190     IHCOND             : TInvestment; //RFPC0018 Conduit
191     IHDS0              : TInvestment; //RFPC0018 DS0
192     IHDS1              : TInvestment; //RFPC0018 DS1
```

```
193     IHDS3              : TInvestment; //RFPC0018 DS3
194     IHSWPORTS          : TInvestment; //RFPC0018 switch ports
195
196     State              : String;      //RFPC0018 State in which CLLI code is located
197     Cur_State          : String;      //RFPC0018 The state the model is being run for
198     WireCtr_CLLI       : String;      //RFPC0018 Associated wire-center for SS7 site
199
200     STPPortInv         : TInvestment; //RFPC0018 Direct investment per STP port
201     STPSpec_CktInv     : TInvestment; //RFPC0018 STP special circuit equipment investment
202     DS01Ratio          : TInvestment; //RFPC0018 channels Ratios
203     DS13Ratio          : TInvestment; //RFPC0018 channels Ratios
204     DSX3DS0            : TInvestment; //RFPC0018 DS0 equivalent channels
205     DSX1DS0            : TInvestment; //RFPC0018 DS0 equivalent channels
206     DCS10DS0           : TInvestment; //RFPC0018 DS0 equivalent channels
207     Clink_CktInv       : TInvestment; //RFPC0018 C link circuit investment
208     CLink_AerFInv      : TInvestment; //RFPC0018 C link aerial fiber investment
209     CLink_BurFInv      : TInvestment; //RFPC0018 C link buried fiber investment
210     CLink_UndFInv      : TInvestment; //RFPC0018 C link underground fiber investment
211     CLink_PoleInv      : TInvestment; //RFPC0018 C link pole investment
212     CLink_CondInv      : TInvestment; //RFPC0018 C link conduit investment
213     STPGS_SwtInv       : TInvestment; //RFPC0018 STP Getting started switch equipment investment
214     STPGS_CktInv       : TInvestment; //RFPC0018 STP Getting started circuit equipment investment
215     STPGS_AerFInv      : TInvestment; //RFPC0018 STP Getting started aerial fiber investment
216     STPGS_BurFInv      : TInvestment; //RFPC0018 STP Getting started buried fiber investment
217     STPGS_UndFInv      : TInvestment; //RFPC0018 STP Getting started underground fiber investment
218     STPGS_PoleInv      : TInvestment; //RFPC0018 STP Getting started pole investment
219     STPGS_CondInv      : TInvestment; //RFPC0018 STP Getting started conduit investment
220     STPGS_Ports        : TInvestment; //RFPC0018 STP Getting started port investment
221     STPPORTSWT         : TInvestment; //RFPC0018 STP port switch equipment investment
222     STPPORTCKT         : TInvestment; //RFPC0018 STP port circuit equipment investment
223     STPPORTAERF        : TInvestment; //RFPC0018 STP port aerial fiber investment
224     STPPORTBURF        : TInvestment; //RFPC0018 STP port buried fiber investment
225     STPPORTUNDF        : TInvestment; //RFPC0018 STP port underground fiber investment
226     STPPORTPOLE        : TInvestment; //RFPC0018 STP port pole investment
227     STPPORTCOND        : TInvestment; //RFPC0018 STP port conduit investment
228     STPPORTEXP         : TInvestment; //RFPC0018 STP port leased expense
229     GTTInv_L           : TInvestment; //RFPC0018 Local STP Global Title Translation (GTT) software investment
230     TCAPSWL            : TInvestment; //RFPC0018 TCAP message at local STP
```

```
231     GTTInv_R          : TInvestment; //RFPC0018 Regional STP Global Title Translation (GTT) software
232  investment
233     TCAPSWR           : TInvestment; //RFPC0018 TCAP message at regional STP
234     DLINKCKT          : TInvestment; //RFPC0018 D link circuit investment
235     DLINKAERF         : TInvestment; //RFPC0018 D link aerial fiber investment
236     DLINKBURF         : TInvestment; //RFPC0018 D link buried fiber investment
237     DLINKUNDF         : TInvestment; //RFPC0018 D link underground fiber investment
238     DLINKPOLE         : TInvestment; //RFPC0018 D link pole investment
239     DLINKCOND         : TInvestment; //RFPC0018 D link conduit investment
240     DLINKEXP          : TInvestment; //RFPC0018 D link leased expense
241     DB800LINKCKT      : TInvestment; //RFPC0018 Database 800 link circuit investment
242     LIDBLINKCKT       : TInvestment; //RFPC0018 LIDB link circuit investment
243     DB800LINKEXP      : TInvestment; //RFPC0018 Database 800 link leased expense
244     LIDBLINKEXP       : TInvestment; //RFPC0018 LIDB link leased expense
245     ALINK800CKT       : TInvestment; //RFPC0018 Database 800 A link circuit investment
246     ALINK800AERF      : TInvestment; //RFPC0018 Database 800 A link aerial fiber investment
247     ALINK800BURF      : TInvestment; //RFPC0018 Database 800 A link buried fiber investment
248     ALINK800UNDF      : TInvestment; //RFPC0018 Database 800 A link underground fiber investment
249     ALINK800POLE      : TInvestment; //RFPC0018 Database 800 A link pole investment
250     ALINK800COND      : TInvestment; //RFPC0018 Database 800 A link conduit investment
251     ALINKLIDBCKT      : TInvestment; //RFPC0018 LIDB A link circuit investment
252     ALINKLIDBAERF     : TInvestment; //RFPC0018 LIDB A link aerial fiber investment
253     ALINKLIDBBURF     : TInvestment; //RFPC0018 LIDB A link buried fiber investment
254     ALINKLIDBUNDF     : TInvestment; //RFPC0018 LIDB A link underground fiber investment
255     ALINKLIDBPOLE     : TInvestment; //RFPC0018 LIDB A link pole investment
256     ALINKLIDBCOND     : TInvestment; //RFPC0018 LIDB A link conduit investment
257     RSPORTSWT         : TInvestment; //RFPC0018 Regional STP Port switch investment
258     RSPORTCKT         : TInvestment; //RFPC0018 Regional STP Port circuit investment
259     RSPORTAERF        : TInvestment; //RFPC0018 Regional STP Port aerial fiber investment
260     RSPORTBURF        : TInvestment; //RFPC0018 Regional STP Port buried fiber investment
261     RSPORTUNDF        : TInvestment; //RFPC0018 Regional STP Port underground fiber investment
262     RSPORTPOLE        : TInvestment; //RFPC0018 Regional STP Port pole investment
263     RSPORTCOND        : TInvestment; //RFPC0018 Regional STP Port conduit investment
264     DB800PORTSWT      : TInvestment; //RFPC0018 Database 800 STP Port switch investment
265     DB800PORTCKT      : TInvestment; //RFPC0018 Database 800 STP Port circuit investment
266     DB800PORTAERF     : TInvestment; //RFPC0018 Database 800 STP Port aerial fiber investment
267     DB800PORTBURF     : TInvestment; //RFPC0018 Database 800 STP Port buried fiber investment
268     DB800PORTUNDF     : TInvestment; //RFPC0018 Database 800 STP Port underground fiber investment
269     DB800PORTPOLE     : TInvestment; //RFPC0018 Database 800 STP Port pole investment
```

```
270     DB800PORTCOND      : TInvestment; //RFPC0018 Database 800 STP Port conduit investment
271     LIDBPORTSWT        : TInvestment; //RFPC0018 LIDB STP Port switch investment
272     LIDBPORTCKT        : TInvestment; //RFPC0018 LIDB STP Port circuit investment
273     LIDBPORTAERF       : TInvestment; //RFPC0018 LIDB STP Port aerial fiber investment
274     LIDBPORTBURF       : TInvestment; //RFPC0018 LIDB STP Port buried fiber investment
275     LIDBPORTUNDF       : TInvestment; //RFPC0018 LIDB STP Port underground fiber investment
276     LIDBPORTPOLE       : TInvestment; //RFPC0018 LIDB STP Port pole investment
277     LIDBPORTCOND       : TInvestment; //RFPC0018 LIDB STP Port conduit investment
278     SSPPORT            : TInvestment; //RFPC0018 SSP Port switch investment
279     ALINKS             : TInvestment; //RFPC0018 SSP A link
280     ALINKCKT           : TInvestment; //RFPC0018 SSP A link circuit investment
281     ALINKEXP           : TInvestment; //RFPC0018 SSP A link leased expense
282     AlinkDistance      : TInvestment; //RFPC0018 SSP Sum of the two A link distances
283     SSP800PORT         : TInvestment; //RFPC0018 SSP database 800 port switch investment
284     SSPLIDBPORT        : TInvestment; //RFPC0018 SSP LIDB port switch investment
285     DSODistance        : TInvestment; //RFPC0018 Weighted DSO distance
286
287   procedure Tf_ss7.Entry;
288   begin
289    // Set forms attributes and make it visible
290     dbg_SS7.Visible  := True;
291     dbn_SS7.Visible  := True;
292     cb_Assoc.Visible := True;
293     p_Status.Visible := False;
294     Width  := 435;
295     Height := 300;
296     Show;
297   end;
298
299   procedure Tf_ss7.SetCaption(TableIndex : Integer);
300   begin
301     Case TableIndex of  //RFPC0018 Set form caption
302       0 : Caption := 'Signaling System No.7 [' + dm_ICM.t_SS7_SSP.TableName+']';        //RFPC0018 State SSP
303   table
304       1 : Caption := 'Signaling System No.7 [' + dm_ICM.t_SS7_STP.TableName+']';        //RFPC0018 STP table
305       2 : Caption := 'Signaling System No.7 [' + dm_ICM.t_SS7_Location.TableName+']';   //RFPC0018 STP
306   location table
307       3 : Caption := 'Signaling System No.7 [' + dm_ICM.t_SS7_Investment.TableName+']'; //RFPC0018 SS7
308   hardware and software investment table
```

```
309     4 : Caption := 'Signaling System No.7 [' + dm_ICM.t_SS7_Parameter.TableName+']';  //RFPC0018 SS7 network
310  parameters table
311     5 : Caption := 'Signaling System No.7 [' + dm_ICM.t_SS7_Loading.TableName+']';    //RFPC0018 SS7 Loading
312  factor table
313    end;
314  end;
315
316  procedure Tf_ss7.OpenFiles(Proc : Boolean);
317  begin
318    Application.ProcessMessages;
319    if Proc
320      then begin // Open require tables for processing
321             { Open Switch Type Table }
322             TabName :=  f_associate.GetFileName(2001); // get switch type table filename and location using
323  index 2001
324             With dm_ICM.t_Switch do   // set switch table variable name to t_switch and setup table
325               begin
326                 if Active then Close;
327                 DatabaseName := ExtractFilePath(TabName);
328                 TableName := ExtractFilename(TabName);
329                 IndexFieldNames := '';
330                 IndexName := '';
331                 TableType := ttDefault;
332                 Exclusive := true;
333                 Open;
334               end;  // end 'with t_switch do'
335
336             { Open Switch investment Table }
337             TabName :=  f_associate.GetFileName(3003);  // get Switch Investment table filename and location
338  using index 3003
339             With dm_ICM.t_SwCost do
340               begin
341                 if Active then Close;
342                 DatabaseName := ExtractFilePath(TabName);
343                 TableName := ExtractFilename(TabName);
344                 TableType := ttDefault;
345                 Exclusive := True;
346                 Open;
347                 IndexName := '';
```

```
348               CancelRange;
349             end;    // end 'with t_swcost do'
350
351           { Nodes Table }
352             TabName :=  f_associate.GetFileName(2003);
353             With dm_ICM.t_Nodes do
354               begin
355                  if Active then Close;
356                  DatabaseName    := ExtractFilePath(TabName);
357                  TableName       := ExtractFilename(TabName);
358                  IndexFieldNames := '';
359                  IndexName       := '';
360                  TableType       := ttDefault;
361                  Exclusive       := True;
362                  Open;
363               end;
364
365           { Host 2 Host Table }
366             With dm_ICM.t_A_H2H do
367               begin
368                  if Active then Close;
369                  DatabaseName    := 'modata';
370                  TableName       := 'acth2h.db';
371                  IndexFieldNames := '';
372                  IndexName       := '';
373                  TableType       := ttDefault;
374                  Exclusive       := True;
375                  Open;
376               end;
377           end;
378
379     TabName :=  f_associate.GetFileName(2010); //RFPC0018 State SSP table
380     if TabName <> ''
381       then begin
382              With dm_ICM.t_SS7_SSP do
383                begin
384                   if Active then Close;
385                   DatabaseName := ExtractFilePath(TabName);
386                   TableName := ExtractFilename(TabName);
```

```
387                IndexFieldNames := '';
388                IndexName := '';
389                TableType := ttDefault;
390                Exclusive := True;
391                Open;
392             end;
393          end;
394
395    TabName :=  f_associate.GetFileName(2017); //RFPC0018 STP table
396    if TabName <> ''
397      then begin
398             With dm_ICM.t_SS7_STP do
399               begin
400                  if Active then Close;
401                  DatabaseName := ExtractFilePath(TabName);
402                  TableName := ExtractFilename(TabName);
403                  IndexFieldNames := '';
404                  IndexName := '';
405                  TableType := ttDefault;
406                  Exclusive := True;
407                  Open;
408               end;
409          end;
410
411    TabName :=  f_associate.GetFileName(2018); //RFPC0018 STP location table
412    if TabName <> ''
413      then begin
414             With dm_ICM.t_SS7_Location do
415               begin
416                  if Active then Close;
417                  DatabaseName := ExtractFilePath(TabName);
418                  TableName := ExtractFilename(TabName);
419                  IndexFieldNames := '';
420                  IndexName := '';
421                  TableType := ttDefault;
422                  Exclusive := True;
423                  Open;
424               end;
425          end;
```

```
426
427     TabName :=  f_associate.GetFileName(2019);//RFPC0018 SS7 hardware and software investment table
428     if TabName <> ''
429       then begin
430              With dm_ICM.t_SS7_Investment do
431                begin
432                   if Active then Close;
433                   DatabaseName := ExtractFilePath(TabName);
434                   TableName := ExtractFilename(TabName);
435                   IndexFieldNames := '';
436                   IndexName := '';
437                   TableType := ttDefault;
438                   Exclusive := True;
439                   Open;
440                end;
441            end;
442
443     TabName :=  f_associate.GetFileName(2020); //RFPC0018 SS7 network parameters table
444     if TabName <> ''
445       then begin
446              With dm_ICM.t_SS7_Parameter do
447                begin
448                   if Active then Close;
449                   DatabaseName := ExtractFilePath(TabName);
450                   TableName := ExtractFilename(TabName);
451                   IndexFieldNames := '';
452                   IndexName := '';
453                   TableType := ttDefault;
454                   Exclusive := True;
455                   Open;
456                end;
457            end;
458
459     TabName :=  f_associate.GetFileName(2021); //RFPC0018 SS7 Loading factor table
460     if TabName <> ''
461       then begin
462              With dm_ICM.t_SS7_Loading do
463                begin
464                   if Active then Close;
```

```
465                 DatabaseName := ExtractFilePath(TabName);
466                 TableName := ExtractFilename(TabName);
467                 IndexFieldNames := '';
468                 IndexName := '';
469                 TableType := ttDefault;
470                 Exclusive := True;
471                 Open;
472               end;
473           end;
474 end;
475
476 procedure Tf_ss7.WriteTemp(CLLI_Code, Item: String; Inv : TInvestment);
477 begin
478  // writes data to a temporary table for later use
479   t_T1_SS7.Insert;
480   t_T1_SS7.FieldByName('CLLI').AsString      := CLLI_Code;
481   t_T1_SS7.FieldByName('Component').AsString := Item;
482   t_T1_SS7.FieldByName('Total Inv').AsFloat  := Inv[1];
483   t_T1_SS7.FieldByName('VS Inv').AsFloat     := Inv[2];
484   t_T1_SS7.FieldByName('VIS Inv').AsFloat    := Inv[3];
485   t_T1_SS7.Post;
486 end;
487
488 procedure Tf_ss7.WriteSS7Act(t_Table : TTable; CLLI_Code, Item, Account: String; Inv : TInvestment);
489 begin
490  // writes data to the output tables, which are rollup for mapping
491   t_Table.Insert;
492   t_Table.FieldByName('CLLI').AsString      := CLLI_Code;
493   t_Table.FieldByName('Account').AsString   := Account;
494   t_Table.FieldByName('Component').AsString := Item;
495   t_Table.FieldByName('Category').AsString  := '7';
496   t_Table.FieldByName('Total Inv').AsFloat  := Inv[1];
497   t_Table.FieldByName('VS Inv').AsFloat     := Inv[2];
498   t_Table.FieldByName('VIS Inv').AsFloat    := Inv[3];
499   t_Table.Post;
500 end;
501
502 procedure Tf_ss7.SS7_Invst_Nodes;
503 Var
```

```
504      I          : Word;
505     CLLI       : String;
506     Size       : Integer;
507     SW_Size    : Integer;
508     SW_Type    : String;
509     State      : String;
510     OK         : Boolean;
511     EFI        : Double;
512     Distance   : Double;
513     TotalDist  : Double;
514     Inv        : TInvestment;
515
516    // This procedure produces SSP investment data. The information is sent to the
517    // temporary table.
518
519  //-------------------------------------------------------------------------------------------
520
521  procedure GetSwitchType;
522  { Procedure GetSwitchType gets a switch type from the Nodes Table (t_nodes) and
523    verifies it as a valid type by searching for the same type in the switch type
524    table (t_switch).  Sets variable OK = False if NOT found. }
525   begin
526     Ok := True;
527     SW_Type := dm_ICM.t_Nodes.FieldByName('Switch Type').AsString;  // get switch type from nodes table
528     if dm_ICM.t_Switch.FindKey([SW_Type]) then // search for the nodes switch type in the switch type table
529       Ok := true
530     else Ok := False;  // nodes switch type not found in switch type table, return False
531
532     //RFPC0060 removed the switch type substitution code, no longer valid
533
534     if Not OK  // if switch type not found, output error message
535       then begin
536              MessageDlg(CLLI+' - Could not find switch type ['+SW_Type+']', mtError, [mbOk],0);
537              WeAreWatching.LogEvent(4,'Switch','Could not find switch type ['+SW_Type+']');
538            end;  // end 'if Not OK'
539  end;  // end GetSwitchType procedure
540
541  //-------------------------------------------------------------------------------------------
542  begin
```

```
543
544     //RFPC0018 Section IV Part 1 Begin
545     {
546        IV.  General Function Definitions
547
548           1.  Investment Table Total Values
549
550               Investment Table values are entered into columns of hardware
551               and software investments, along with quantities of hours for
552               engineering, installation, and installation testing.  These
553               values are used along with the switch node data, loading factors,
554               and labor rates to determine total investment values by item for
555               main processing.
556
557               Following is the algorithm for total investment by item.
558
559           If SS7_Invst.Node_Type  [5ESS, DMS-10, DMS-100, GTD-5], apply the
560           Engineer/Furnish (EF&I-EF) and Install (EF&I-I) factors from the
561           Switching EF&I Table, based upon the line size, as in the Switching
562           Module.
563
564           (Use this algorithm for investments at each individual SSP)
565
566            If SS7_Invst.Node_Type  [5ESS, DMS-10, DMS-100, GTD-5], use the
567            following equation.
568
569           if switch related ( not account 223200) use:
570              Total Investment = (Hardware + Software ) * EF&I
571
572           else use:
573              Total Investment = (Hardware * HW_Factor) + (Software * SW_Factor)
574                               + (Eng_Hours * Eng_Rate) + (Inst_Hours * Inst_Rate)
575                               + (Test_Hours * Test_Rate)
576
577           Hardware and Software are investment values, and Eng_Hours, Inst_Hours,
578           and Mtce_Hours are the hours by item from the SS7 Investment Table.
579           HW_Factor and SW_Factor are the combined supply and minor material
580           loading factors for switching equipment applicable to SSPs only.
581           SW_Factor, Eng_Rate, Inst_Rate, and Mtce_Rate are loaded labor rates
```

```
582           from the SS7 Loading Table.
583     }
584     dm_ICM.t_SS7_SSP.First;           //RFPC0018 Go to the beginning of the SSP table
585     while Not dm_ICM.t_SS7_SSP.EOF do
586       begin
587         CLLI := dm_ICM.t_SS7_SSP.FieldByName('SSP_CLLI').AsString;
588         State := Copy(CLLI, 5, 2);
589         if dm_ICM.t_Nodes.FindKey([CLLI])
590           then begin
591                  GetSwitchType; //RFPC0018 Get Switch Type
592
593                  //RFPC0018 Retrieve EF&I Factor
594                  //RFPC0060 The EFI factor is added to the switch investment table as
595                  // component M0004. Read from swinv.db.
596                  if dm_ICM.t_SwCost.FindKey([CLLI,'M0004']) // find switch record in EFI table
597                    then EFI := dm_ICM.t_SwCost.FieldByName('Average Invest').AsFloat  //
598 engineering/furnishing factors
599                    else begin  // EFI Not Found
600                           EFI := 1.0;  //Set EFI to 1
601                         end;  // end else (EFI not found)
602
603                  if dm_ICM.t_SS7_Loading.FindKey([State])
604                    then begin
605                           dm_ICM.t_SS7_Investment.CancelRange; //RFPC0018 Clear filter on SS7 investment
606 table
607                           dm_ICM.t_SS7_Investment.SetRange(['SSP'], ['SSP']) ; //RFPC0018 Set filter on SS7
608 investment table
609                           dm_ICM.t_SS7_Investment.First;
610                           while Not dm_ICM.t_SS7_Investment.EOF do
611                             begin
612                               if SW_Type = dm_ICM.t_SS7_Investment.FieldByName('Technology').AsString
613                                 then begin
614                                        if dm_ICM.t_SS7_Investment.FieldByName('Account').AsString = '223200'
615                                          then Inv[1] :=
616 (dm_ICM.t_SS7_Investment.FieldByName('Hardware').AsFloat *
617                                                          (1.0 +
618 dm_ICM.t_SS7_Loading.FieldByName('HW_Factor').AsFloat )) +
619                                                          (dm_ICM.t_SS7_Investment.FieldByName('Software').AsFloat
620 *
```

```
621                                                    (1.0 +
622 dm_ICM.t_SS7_Loading.FieldByName('SW_Factor').AsFloat )) +
623                                                    (dm_ICM.t_SS7_Investment.FieldByName('Eng_Hrs').AsFloat
624 *
625                                                     dm_ICM.t_SS7_Loading.FieldByName('Eng_Rate').AsFloat )
626 +
627                                                    (dm_ICM.t_SS7_Investment.FieldByName('Inst_Hrs').AsFloat
628 *
629                                                     dm_ICM.t_SS7_Loading.FieldByName('Inst_Rate').AsFloat )
630 +
631                                                    (dm_ICM.t_SS7_Investment.FieldByName('Test_Hrs').AsFloat
632 *
633                                                     dm_ICM.t_SS7_Loading.FieldByName('Test_Rate').AsFloat )
634                                          else Inv[1] :=
635 (dm_ICM.t_SS7_Investment.FieldByName('Hardware').AsFloat  +
636
637 dm_ICM.t_SS7_Investment.FieldByName('Software').AsFloat) * EFI;
638                                          Inv[2] := Inv[1];
639                                          WriteTemp(CLLI, dm_ICM.t_SS7_Investment.FieldByName('Item').AsString,
640 Inv);
641                                      end;
642                                dm_ICM.t_SS7_Investment.Next;
643                              end;
644                         end
645                      else begin  // State Not Found
646                             MessageDlg('Could not find state for'+ CLLI + '.', mtError, [mbok], 0);
647                             WeAreWatching.LogEvent(1,'Process','SS7 ***ERROR*** SSP CLLI State not found.');
648                           end;  // end else (State not found)
649                end
650             else begin  // CLLI Not Found
651                    EFI := 1.0;  //Set EFI to 1
652                    MessageDlg('Could not find '+ CLLI + ' in nodes table.', mtError, [mbok], 0);
653                    WeAreWatching.LogEvent(1,'Process','SS7 ***ERROR*** SSP CLLI: '
654                              + CLLI + ' not found.');
655                end;  // end else (CLLI not found)
656
657          dm_ICM.t_SS7_SSP.Next;
658        end;
659      //RFPC0018 Section IV Part 1 End
```

```
660
661     //RFPC0018 Section VIII Part 3 Begin
662     {
663         3.  SSP Port Investment
664
665         SSP port investments are stored in the SS7 Investment Table
666         as per port values for the initial pair and for each additional
667         pair.  The investment per port depends upon the number of A
668         links terminating at the SSP.
669
670         SSPPORT = [2 * SSPIPort + ((ALinks - 2) * SSPAPort)] / ALinks
671         where SSPIPort and SSPAPort are the total investments per link
672         pair for the initial pair and for additional SSP ports,
673         respectively, from the SS7 Investment Table, and ALinks is the
674         number of ports used at the SSP, from the SSP Table.
675     }
676      dm_ICM.t_SS7_SSP.First;           //RFPC0018 Go to the beginning of the SSP table
677     while Not dm_ICM.t_SS7_SSP.EOF do
678       begin
679         Application.ProcessMessages;
680       //RFPC0018 initialize values
681         for I := 1 to 2 do
682            begin
683              SSPPORT[I]    := 0.0;
684              ALINKCKT[I]   := 0.0;
685              ALINKEXP[I]   := 0.0;
686              AlinkDistance[I] := 0.0;
687              SSP800PORT[I]:= 0.0;
688              SSPLIDBPORT[I]:=0.0;
689            end;
690
691         CLLI := dm_ICM.t_SS7_SSP.FieldByName('SSP_CLLI').AsString;
692         //RFPC0018 A Links
693         ALinks[1] := dm_ICM.t_SS7_SSP.FieldByName('ALink').AsFloat;
694         ALinks[2] := ALinks[1];
695         ALinks[3] := ALinks[1];
696
697         //RFPC0018 calculate SSP port
698         if GetSS7Temp(CLLI, 'SSPAPort')
```

```
699           then begin
700                 SSPPORT[1] := SSPPORT[1] + (dm_ICM.t_SS7_SSP.FieldByName('ALink').AsFloat - 2.0) *
701                               t_T1_SS7.FieldByName('Total Inv').AsFloat;
702                 SSPPORT[2] := SSPPORT[2] + (dm_ICM.t_SS7_SSP.FieldByName('ALink').AsFloat - 2.0) *
703                               t_T1_SS7.FieldByName('VS Inv').AsFloat;
704               end;
705         if GetSS7Temp(CLLI, 'SSPIPort')
706           then begin
707                 SSPPORT[1] := (SSPPORT[1] + t_T1_SS7.FieldByName('Total Inv').AsFloat * 2.0) /
708                               dm_ICM.t_SS7_SSP.FieldByName('ALink').AsFloat;
709                 SSPPORT[2] := (SSPPORT[2] + t_T1_SS7.FieldByName('VS Inv').AsFloat * 2.0) /
710                               dm_ICM.t_SS7_SSP.FieldByName('ALink').AsFloat;
711               end;
712     //RFPC0018 Section VII Part 3 End
713
714     //RFPC0018 Section VII Part 4 begin
715     {
716
717           4.  Investment per A Link
718
719           A link investments should be calculated individually for links
720           to Local STP1 and to Local STP 2 (defined in the SSP Table as
721           Local_STP1 and Local_STP2, respectively), with results for the
722           two routes added within each plant account.
723
724         Circuit Equipment Investment:
725
726           ALINKCKT is the investment per A link for circuit equipment.
727
728           First calculate special circuit investment per link:
729
730           SSPSpec_CktInv = Total investment value for "SSP_SPEC_CKT" from the SS7
731           Investment Table for SS7_Invst.Switch_type = SSPVendor
732           Use the investment and unit values for the SSP from the Acth2h Transport
733           Module output file, and use the total investment values for special
734           circuit terminations, SSP_SPEC_CKT and STP_SPEC_CKT, from the SS7
735           Investment Table.
736
737           Ratio of Channel Bank DS1 terminations to total DSX1 terminations, and
```

```
738           ratio of DSX1 DS3 terminations to total DS3s:      (Using 24 DS0s per DS1
739           and 28 DS1s per DS3)
740
741           DS01Ratio = (Round Up (IHCBDS0/24) ) / IHDSX1DS1
742           DS13Ratio = (Round Up (IHDSX1DS1/28) ) / IHDSX3DS3
743
744           ALINKCKT = (2 * SSPSpec_CktInv)  + STPSpec_Ckt for local STP 1
745                       + STPSpec_Ckt for local STP 2
746
747
748     }
749         State        := Copy(CLLI, 5, 2);                                    //RFPC0018 Extract state code
750         WireCtr_CLLI := CLLI;
751         GetInterOfficeValues;                                                    //RFPC0018 Get inter-
752   office values
753
754         //RFPC0018 Calc DS01Ratio
755         for I := 1 to 2 do
756          if IHDSX1DS1[I] > 0.0
757            then DS01Ratio[I] := Ceil(IHCBDS0[I] / 24.0) / IHDSX1DS1[I]
758            else DS01Ratio[I] := 0.0;
759
760         //RFPC0018 Calc DS13Ratio
761         for I := 1 to 2 do
762         if IHDSX3DS3[I] > 0.0
763            then  DS13Ratio[I] := Ceil(IHDSX1DS1[I] / 28.0) / IHDSX3DS3[I]
764            else  DS13Ratio[I] := 0.0;
765
766         //RFPC0018 calculate A link circuit
767         if GetSS7Temp(dm_ICM.t_SS7_SSP.FieldByName('Local_STP1').AsString, 'STPSpec_CktInv')
768           then begin
769                  ALINKCKT[1] := ALINKCKT[1] + t_T1_SS7.FieldByName('Total Inv').AsFloat;
770                  ALINKCKT[2] := ALINKCKT[2] + t_T1_SS7.FieldByName('VS Inv').AsFloat;
771                end;
772         if GetSS7Temp(dm_ICM.t_SS7_SSP.FieldByName('Local_STP2').AsString, 'STPSpec_CktInv')
773           then begin
774                  ALINKCKT[1] := (ALINKCKT[1] + t_T1_SS7.FieldByName('Total Inv').AsFloat);
775                  ALINKCKT[2] := (ALINKCKT[2] + t_T1_SS7.FieldByName('VS Inv').AsFloat);
776                end;
```

```
777         if GetSS7Temp(CLLI, 'SSP_SPEC_CKT')
778           then begin
779                  ALINKCKT[1] := ALINKCKT[1] + t_T1_SS7.FieldByName('Total Inv').AsFloat * 2.0;
780                  ALINKCKT[2] := ALINKCKT[2] + t_T1_SS7.FieldByName('VS Inv').AsFloat * 2.0;
781                end;
782
783       {RFPC0018 expense
784
785            5. Expense per A Link
786
787               Expense result for the SSP is the sum of results for individual
788               links (to Local STP 1 and to Local STP 2).  This expense is used
789               only for retail costs.
790
791            ALINKEXP = Monthly Lease Expense = A1_Expense and/or A2_Expense,
792            where A1_Expense and A2_Expense are the monthly expenses from the
793            respective columns of the STATE_SSP Table.
794       }
795       if (dm_ICM.t_SS7_SSP.FieldByName('A1_Expense').AsFloat > 0.0)
796        then ALINKEXP[1] := dm_ICM.t_SS7_SSP.FieldByName('A1_Expense').AsFloat;
797
798       if (dm_ICM.t_SS7_SSP.FieldByName('A2_Expense').AsFloat > 0.0)
799        then ALINKEXP[1] := ALINKEXP[1] + dm_ICM.t_SS7_SSP.FieldByName('A2_Expense').AsFloat;
800
801       ALINKEXP[2] := ALINKEXP[1];
802
803       {RFPC0018 outside plant calculations
804
805            Compute distance between the SSP and each of the STPs.
806
807            Latitude/Longitude Coordinates for the Local STPs:
808
809            Lat_STP1 = STP_Table.Latitude where STP_Table.STP_CLLI = Local_STP1
810            Long_STP1 = STP_Table.Longitude where STP_Table.STP_CLLI = Local_STP1
811
812            Lat_STP2 = STP_Table.Latitude where STP_Table.STP_CLLI = Local_STP2
813            Long_STP2 = STP_Table.Longitude where STP_Table.STP_CLLI = Local_STP2
814
815            A Link Distances:
```

```
816
817             Set ALMRATIO, the air mileage to route mileage ratio.
818
819             ADistance1 = ALMRATIO * Distance(Latitude, Longitude, LatSTP1, LongSTP1)
820             where Latitude and Longitude are the coordinates of this SSP from the
821             Node Table.
822
823             ADistance2 = ALMRATIO * Distance(Latitude, Longitude, LatSTP2, LongSTP2)
824             where Latitude and Longitude are the coordinates of this SSP from
825             the Node Table.
826         }
827
828         Distance := 0.0;
829         //RFPC0018 find latitude and longitude for ssp
830         if dm_ICM.t_Nodes.FindKey([CLLI])
831           then begin
832                 //RFPC0018 find latitude and longitude for stp1
833                 if dm_ICM.t_SS7_STP.FindKey([dm_ICM.t_SS7_SSP.FieldByName('Local_STP1').AsString])
834                    then begin
835                           if (dm_ICM.t_SS7_SSP.FieldByName('A1_Expense').AsFloat <= 0.0001)
836                             then Distance := CalcDistance(
837                                    dm_ICM.t_Nodes.FieldByName('Latitude').AsFloat,
838                                    dm_ICM.t_Nodes.FieldByName('Longitude').AsFloat,
839                                    dm_ICM.t_SS7_STP.FieldByName('STP_Latitude').AsFloat,
840                                    dm_ICM.t_SS7_STP.FieldByName('STP_Longitude').AsFloat);
841                         end
842                    else begin  // STP Not Found
843                           MessageDlg('STP Not Found For '+ CLLI + '.', mtError, [mbok], 0);
844                           WeAreWatching.LogEvent(1,'Process','SS7 ***ERROR*** SSP CLLI STP Not Found for '+
845 CLLI);
846                         end;  // end else (STP Not Found)
847
848                 //RFPC0018 find latitude and longitude for stp2
849                 if dm_ICM.t_SS7_STP.FindKey([dm_ICM.t_SS7_SSP.FieldByName('Local_STP2').AsString])
850                   then begin
851                          if (dm_ICM.t_SS7_SSP.FieldByName('A2_Expense').AsFloat <= 0.0001)
852                            then Distance := Distance + CalcDistance(
853                                    dm_ICM.t_Nodes.FieldByName('Latitude').AsFloat,
854                                    dm_ICM.t_Nodes.FieldByName('Longitude').AsFloat,
```

```
                                    dm_ICM.t_SS7_STP.FieldByName('STP_Latitude').AsFloat,
                                    dm_ICM.t_SS7_STP.FieldByName('STP_Longitude').AsFloat);
                        end
                    else begin  // STP Not Found
                          MessageDlg('STP Not Found For '+ CLLI + '.', mtError, [mbok], 0);
                          WeAreWatching.LogEvent(1,'Process','SS7 ***ERROR*** SSP CLLI STP Not Found for '+
CLLI);
                        end;  // end else (STP Not Found)

              end
         else begin  // STP Not Found
                MessageDlg('CLLI not found in node table. '+ CLLI + '.', mtError, [mbok], 0);
                WeAreWatching.LogEvent(1,'Process','SS7 ***ERROR*** SSP CLLI not found in node table. '+
CLLI);
              end;  // end else (STP Not Found)

       Distance := GetParameter('ALMRATIO') * Distance / 5280.0;
       AlinkDistance[1] := Distance;
       AlinkDistance[2] := Distance;
       AlinkDistance[3] := Distance;

    //RFPC0018 Section VII Part 4 End

    //RFPC0018 Section VII Part 6 begin

       //RFPC0018 calculate software investments
    {
         6.  SSP DB SoftwareInvestments per Port

             Software is required at each SSP for database query services.  These
             Right to Use (RTU) fees are stored as totals for the SSP and are
             converted to software investments per port using link quantities.
             The software investments per port are later used to compute software
             investments per query.

           SSP800PORT = SSP800_SW / ALinks
           SSPLIDBPORT = SSPLIDB_SW / ALinks
           SSPLNPPORT = SSPLNP_SW / ALinks
    }
```

```
894         if GetSS7Temp(CLLI, 'SSP800_SW')
895           then begin
896                  SSP800PORT[1] := t_T1_SS7.FieldByName('Total Inv').AsFloat /
897                                   dm_ICM.t_SS7_SSP.FieldByName('ALink').AsFloat;
898                  SSP800PORT[2] := t_T1_SS7.FieldByName('VS Inv').AsFloat /
899                                   dm_ICM.t_SS7_SSP.FieldByName('ALink').AsFloat;
900                end;
901
902         if GetSS7Temp(CLLI, 'SSPLIDB_SW')
903           then begin
904                  SSPLIDBPORT[1] := t_T1_SS7.FieldByName('Total Inv').AsFloat /
905                                    dm_ICM.t_SS7_SSP.FieldByName('ALink').AsFloat;
906                  SSPLIDBPORT[2] := t_T1_SS7.FieldByName('VS Inv').AsFloat /
907                                    dm_ICM.t_SS7_SSP.FieldByName('ALink').AsFloat;
908                end;
909       //RFPC0018 Section VII Part 6 End
910
911       //RFPC0018 write data out to a temporary table
912          WriteTemp(CLLI, 'ALINKS',         ALinks );
913          WriteTemp(CLLI, 'SSPPORT',        SSPPORT );
914          WriteTemp(CLLI, 'ALINKCKT',       ALINKCKT );
915          WriteTemp(CLLI, 'ALINKEXP',       ALINKEXP);
916          WriteTemp(CLLI, 'ALINKDISTANCE', AlinkDistance);
917          WriteTemp(CLLI, 'SSP800PORT',     SSP800PORT );
918          WriteTemp(CLLI, 'SSPLIDBPORT',    SSPLIDBPORT);
919
920          dm_ICM.t_SS7_SSP.Next;
921        end;
922   end;
923
924   procedure Tf_ss7.SS7_Invst_STP;
925   Var
926     CurrentRec: TBookmark;
927     CLLI      : String;
928     RCLLI     : String;
929     SW_Type   : String;
930     Inv       : TInvestment;
931     I         : Integer;
932     X         : Double;
```

```
933     CLink_Expense : Double;
934     STPGSExp  : Double;
935     Distance  : Double;
936     TotalDist : Double;
937     GTTMsgCap : Double;
938     GTTMsgDem : Double;
939
940    // This procedure produces STP investment data. The information is sent to the
941    // temporary table.
942   begin
943     //RFPC0018 Section VI Part 1 Begin
944     {
945          Compute Total Loaded Investment for items at the STP .
946
947          Total Investment = (Hardware * HW_Factor) + (Software * SW_Factor)
948                           + (Eng_Hours * Eng_Rate) + (Inst_Hours * Inst_Rate)
949                           + (Test_Hours * Test_Rate)
950     }
951     dm_ICM.t_SS7_STP.First;           //RFPC0018 Go to the beginning of the STP table
952     while Not dm_ICM.t_SS7_STP.EOF do
953       begin
954         Application.ProcessMessages;
955         CLLI    := dm_ICM.t_SS7_STP.FieldByName('STP_CLLI').AsString;
956         SW_Type := dm_ICM.t_SS7_STP.FieldByName('STP_Type').AsString;
957         State := Copy(CLLI, 5, 2);
958
959         if dm_ICM.t_SS7_Loading.FindKey([State])
960           then begin
961                 dm_ICM.t_SS7_Investment.CancelRange; //RFPC0018 Clear filter on SS7 investment table
962                 dm_ICM.t_SS7_Investment.SetRange(['STP'], ['STP']) ; //RFPC0018 Set filter on SS7 investment
963   table
964                 dm_ICM.t_SS7_Investment.First;
965                 while Not dm_ICM.t_SS7_Investment.EOF do
966                   begin
967                     if SW_Type = dm_ICM.t_SS7_Investment.FieldByName('Technology').AsString
968                       then begin
969                             Inv[1] := (dm_ICM.t_SS7_Investment.FieldByName('Hardware').AsFloat *
970                                     (1.0 + dm_ICM.t_SS7_Loading.FieldByName('HW_Factor').AsFloat )) +
971                                    (dm_ICM.t_SS7_Investment.FieldByName('Software').AsFloat *
```

```
                                        (1.0 + dm_ICM.t_SS7_Loading.FieldByName('SW_Factor').AsFloat )) +
                                       (dm_ICM.t_SS7_Investment.FieldByName('Eng_Hrs').AsFloat *
                                        dm_ICM.t_SS7_Loading.FieldByName('Eng_Rate').AsFloat ) +
                                       (dm_ICM.t_SS7_Investment.FieldByName('Inst_Hrs').AsFloat *
                                        dm_ICM.t_SS7_Loading.FieldByName('Inst_Rate').AsFloat ) +
                                       (dm_ICM.t_SS7_Investment.FieldByName('Test_Hrs').AsFloat *
                                        dm_ICM.t_SS7_Loading.FieldByName('Test_Rate').AsFloat );
                              Inv[2] := Inv[1];

                              WriteTemp(CLLI, dm_ICM.t_SS7_Investment.FieldByName('Item').AsString,  Inv);
                            end;
                        dm_ICM.t_SS7_Investment.Next;
                      end;
                end
            else begin  // State Not Found
                  MessageDlg('Could not find state for'+ CLLI + '.', mtError, [mbok], 0);
                  WeAreWatching.LogEvent(1,'Process','SS7 ***ERROR*** SSP CLLI State not found.');
                end;  // end else (State not found)

        dm_ICM.t_SS7_STP.Next;
      end;
    //RFPC0018 Section VI Part 1 End

    dm_ICM.t_SS7_STP.First;           //RFPC0018 Go to the beginning of the STP table
    while Not dm_ICM.t_SS7_STP.EOF do
      begin
        CLLI         := dm_ICM.t_SS7_STP.FieldByName('STP_CLLI').AsString;     //RFPC0018 Get Clli Code
        SW_Type      := dm_ICM.t_SS7_STP.FieldByName('STP_Type').AsString;     //RFPC0018 Get equipment type
        State        := Copy(CLLI, 5, 2);                                      //RFPC0018 Extract state code
        WireCtr_CLLI := dm_ICM.t_SS7_STP.FieldByName('WCenter_CLLI').AsString; //RFPC0018 Get equipment type
        GetInterOfficeValues;                                                       //RFPC0018 Get inter-
office values

    //RFPC0018 Section VI Part 2 begin
     {
          2. Direct Investment per Port

          Following is the equation for the direct investment per port.  The total
          investment per port will be determined later to include STP common costs.
```

```
1011
1012          Let STPMtceGrowth = Value from Network Parameters Table and STPPort = Total
1013          Investment value for an STP Port where SS7_Invst.Node_Type = STP_Table.STP_Type.
1014
1015          STPPortInv = STPPort * (1 + STPMtceGrowth)
1016       }
1017          if GetSS7Temp(CLLI, 'STPPort')
1018            then begin
1019                   X := GetParameter('STPMtceGrowth');
1020                   STPPortInv[1] := t_T1_SS7.FieldByName('Total Inv').AsFloat * (1.0 + X);
1021                   STPPortInv[2] := t_T1_SS7.FieldByName('VS Inv').AsFloat * (1.0 +  X);
1022                 end;
1023       //RFPC0018 Section VI Part 2 end
1024
1025       //RFPC0018 Section VI Part 3 begin
1026       {
1027            3. Investment per C Link, by STP
1028
1029            Circuit equipment investments are required for any link terminating at
1030            the STP, and outside plant investments are required for the C links
1031            between the mated STPs.
1032
1033            Circuit Equipment Investment:
1034
1035            CLink_CktInv is the investment per C link for circuit equipment.
1036
1037            First calculate special circuit investment per link:
1038
1039            STPSpec_CktInv = Total investment value for item "STP_SPEC_CKT" from the
1040            SS7 Investment Table where SS7_Invst.Node_Type = STP_Table.STP_Type.
1041
1042            If the STP is in the state being studied, based on SS7_Invst.State,
1043            then use the transport investments and units for that STP, using the
1044            values in the Acth2h table for CLLI Code = SS7_Invst.WCenter_CLLI.
1045            Otherwise, use the state-total Transport Module outputs computed in
1046            Section V.
1047       }
1048          if GetSS7Temp(CLLI, 'STP_SPEC_CKT')
1049            then begin
```

```
1050                STPSpec_CktInv[1] := t_T1_SS7.FieldByName('Total Inv').AsFloat;
1051                STPSpec_CktInv[2] := t_T1_SS7.FieldByName('VS Inv').AsFloat;
1052              end;
1053    {  RFPC0018
1054       DSX3DS0  = 24.0 * IHSWPORTS + IHCBDS0 + 24.0 * IHDS1 + 672.0 * IHDS3;
1055       DSX1DS0  = 24.0 * IHSWPORTS + IHCBDS0 + 24.0 * IHDS1;
1056       DCS10DS0] = IHCBDS0 + 24.0 * IHDS1;
1057
1058      Clink_CktInv[I] := STPSpec_CktInv[I] + (24.0 / CLinks) * ((IHFDP[I]+IHADM[I]
1059                         +IHDSX3[I])/DSX3DS0[I] + (IHDCS31[I]+IHMUX31[I]+IHDSX1[I])
1060                         /DSX1DS0[I] + IHDCS10 / DCS10DS0 ) + IHCB[I] / IHCBDS0,
1061                         using investments and units from table "Acth2h".
1062    }
1063        //RFPC0018 Calc DSX3DS0, DSX1DS0 and DCS10DS0
1064        for I := 1 to 2 do
1065         begin
1066           DSX3DS0[I]  := 24.0 * IHSWPORTS[I] + IHCBDS0[I] + 24.0 * IHDS1[I] + 672.0 * IHDS3[I];
1067           DSX1DS0[I]  := 24.0 * IHSWPORTS[I] + IHCBDS0[I] + 24.0 * IHDS1[I];
1068           DCS10DS0[I] := IHCBDS0[I] + 24.0 * IHDS1[I];
1069         end;
1070
1071        //RFPC0018 Calc Clink_CktInv
1072        for I := 1 to 2 do
1073          if IHCBDS0[I] > 0.0
1074           then Clink_CktInv[I] := STPSpec_CktInv[I] +
1075                                   ( (IHFDP[I]+IHADM[I]+IHDSX3[I])/DSX3DS0[I] +
1076                                     (IHDCS31[I]+IHMUX31[I]+IHDSX1[I])/DSX1DS0[I] +
1077                                      IHDCS10[I] / DCS10DS0[I]
1078                                   ) + IHCB[I] / IHCBDS0[I]
1079           else Clink_CktInv[I] := 0.0;
1080
1081        if dm_ICM.t_SS7_STP.FieldByName('CLink_Expense').AsFloat > 0.0
1082          then begin //RFPC0018 CLinks are leased
1083                  for I := 1 to 2 do
1084                   begin
1085                     CLink_AerFInv[I] := 0.0;
1086                     CLink_BurFInv[I] := 0.0;
1087                     CLink_UndFInv[I] := 0.0;
1088                     CLink_PoleInv[I] := 0.0;
```

```
1089                    CLink_CondInv[I] := 0.0;
1090                  end;
1091
1092                  //RFPC0018 Section VI part 4 - Load clink monthly expense
1093    {
1094          4. Expense per C Link, by STP
1095
1096            CLink_Expense = STP_Table.CLink_Expense for this STP_CLLI.
1097            (Monthly expense)
1098    }
1099               CLink_Expense := dm_ICM.t_SS7_STP.FieldByName('CLink_Expense').AsFloat;
1100
1101                //RFPC0018 Section VI part 5 - assigned monthly lease expense
1102                STPGSExp := CLink_Expense * (dm_ICM.t_SS7_STP.FieldByName('CLinks').AsFloat / 2.0);
1103             end
1104
1105          else begin //RFPC0018 Calculate CLinks
1106                 CLink_Expense := 0.0;
1107                 STPGSExp := 0.0;
1108    {
1109
1110          Following are the investment terms for each account of outside plant equipment.
1111
1112          CLink_AerFInv
1113          CLink_BurFInv
1114          CLink_UndFInv
1115          CLink_PoleInv
1116          CLink_CondInv
1117
1118          Outside plant investment calculations depend on whether the C link is
1119          made up of leased or GTE facilities, then on whether STP-specific data
1120          is available in the STP Table.  If STP_Table.CLink_Expense > 0 for this
1121          STP_CLLI, then set all of the above values to 0 and proceed to Lease
1122          Expense Calculations.
1123
1124          GTE Circuit Case - C Link Distance
1125
1126          The length of the C link, in miles, is computed as an airline distance.
1127          First determine latitude and longitude of the mate STP:
```

```
1128
1129          Lat_MateSTP = STP_Table.Latitude and
1130          Long_MateSTP = STP_Table.Longitude
1131
1132          where STP_Table.STP_CLLI = STATE_SSP.Local_STP2 if this STP_CLLI =
1133          Local_STP1 and STP_Table.STP_CLLI = STATE_SSP.Local_STP1 if this
1134          STP_CLLI = Local_STP2 (for a given SSP_CLLI)
1135
1136          Now let ALMRATIO, the route mile to airline mile ratio.
1137          Then calculate C Link distance using the distance function and the
1138          coordinates:
1139
1140          CDistance = ALMRATIO * Distance(Latitude, Longitude, Lat_MateSTP,
1141                      Long_MateSTP) where Latitude and Longitude are the
1142                      coordinates of the given STP_CLLI.
1143
1144          If the STP is in the state being studied, based on SS7_Invst.State,
1145          then use the transport investments and units ("IH…") for that STP,
1146          using the values in the Acth2h table for CLLI Code = SS7_Invst.WCenter_CLLI.
1147          Otherwise, use the state-total Transport Module outputs computed in Section V.
1148
1149          Let TOTALDIST = IHAERDIST + IHBURDIST + IHUNGDIST,
1150          where IHAERDIST, IHBURDIST, and IHUNGDIST are the route distance values
1151          for aerial, buried, and underground cable, respectively, from the table
1152          "Acth2h"
1153
1154          CLink_AerFInv = (IHAERFI / DSODISTANCE)* CDistance
1155
1156          CLink_BurFInv = (IHBURFI / DSODISTANCE) * CDistance
1157
1158          CLink_UndFInv = (IHUNDFI / DSODISTANCE) * CDistance
1159
1160          CLink_PoleInv = (IHPOLES / DSODISTANCE) * CDistance
1161
1162          CLink_CondInv = (IHCOND / DSODISTANCE) * CDistance
1163      }
1164                X := GetParameter('ALMRATIO');
1165                Distance := dm_ICM.t_SS7_STP.FieldByName('CLink_Distance').AsFloat * X;
1166
```

```
1167                for I := 1 to 2 do
1168                 if DSODistance[I] > 0.0
1169                   then begin
1170                         //RFPC0018 Calculate the cLink's aerial cable invesment
1171                         Clink_AerFInv[I] := IHAERFI[I] / DSODistance[I] * Distance;
1172                         //RFPC0018 Calculate the cLink's buried cable invesment
1173                         Clink_BurFInv[I] := IHBURFI[I] / DSODistance[I] * Distance;
1174                         //RFPC0018 Calculate the cLink's underground cable invesment
1175                         Clink_UndFInv[I] := IHUNDFI[I] / DSODistance[I] * Distance;
1176                         //RFPC0018 Calculate the cLink's poles invesment
1177                         Clink_PoleInv[I] := IHPOLES[I] / DSODistance[I] * Distance;
1178                         //RFPC0018 Calculate the cLink's conduit invesment
1179                         Clink_CondInv[I] := IHCOND[I] / DSODistance[I] * Distance;
1180                       end
1181                   else begin
1182                         Clink_AerFInv[I] := 0.0;
1183                         Clink_BurFInv[I] := 0.0;
1184                         Clink_UndFInv[I] := 0.0;
1185                         Clink_PoleInv[I] := 0.0;
1186                         Clink_CondInv[I] := 0.0;
1187                       end;
1188             end;
1189    //RFPC0018 Section VI Part 3 end
1190
1191    //RFPC0018 Section VI Part 5 begin
1192
1193        //RFPC0018 Get Testports
1194        X := GetParameter('TestPorts');
1195
1196        //RFPC0018 Calculate STP getting started investment
1197    {
1198          5. STP Getting Started Investment
1199
1200            The Getting Started Investment per STP includes common equipment,
1201            four ports for testing, between two and eight ports for C links,
1202            link investments for GTE-owned C Link routes, and expenses for
1203            leased C Link routes.
1204
1205            Use TestPorts from the Network Parameters Table and CLinks from the
```

```
1206            STP Table.
1207
1208            STP Getting Started Investment by equipment category:
1209
1210            Switching Equipment:
1211
1212            STPGS_Swt = {[STPCommon + STPPortInv * (TestPorts + CLinks)]
1213
1214            where STPCommon is the total from the SS7 Investment Table for the
1215            appropriate node type, and STPPortInv.
1216
1217
1218         Circuit Equipment:
1219
1220              STPGS_CktInv  = CLinks * CLink_CktInv
1221     }
1222         //RFPC0018 Common
1223
1224         if GetSS7Temp(CLLI, 'STPCommon')
1225           then begin
1226                  STPGS_SwtInv[1] := t_T1_SS7.FieldByName('Total Inv').AsFloat + STPPortInv[1] *
1227                                     (X + dm_ICM.t_SS7_STP.FieldByName('CLinks').AsFloat);
1228                  STPGS_SwtInv[2] := t_T1_SS7.FieldByName('VS Inv').AsFloat + STPPortInv[2] *
1229                                     (X + dm_ICM.t_SS7_STP.FieldByName('CLinks').AsFloat);
1230                end;
1231
1232         //RFPC0018 Circuit
1233         for I := 1 to 2 do STPGS_CktInv[I] := dm_ICM.t_SS7_STP.FieldByName('CLinks').AsFloat *
1234   Clink_CktInv[I];
1235
1236     {
1237        Outside Plant Investments:
1238
1239        (For outside plant, number of C links (equal to CLinks) is divided by 2 so
1240         that half of the investment is assigned to each STP of the mated pair)
1241
1242         STPGS_AerFInv  = (CLinks / 2) * CLink_AerFInv
1243         STPGS_BurFInv  = (CLinks / 2) * CLink_BurFInv
1244         STPGS_UndFInv  = (CLinks / 2) * CLink_UndFInv
```

```
1245        STPGS_PoleInv  = (CLinks / 2) * CLink_PoleInv
1246        STPGS_CondInv  = (CLinks / 2) * CLink_CondInv
1247    }
1248        //RFPC0018 Links
1249        for I := 1 to 2 do STPGS_AerFInv[I] := dm_ICM.t_SS7_STP.FieldByName('CLinks').AsFloat / 2.0 *
1250  Clink_AerFInv[I];
1251        for I := 1 to 2 do STPGS_BurFInv[I] := dm_ICM.t_SS7_STP.FieldByName('CLinks').AsFloat / 2.0 *
1252  Clink_BurFInv[I];
1253        for I := 1 to 2 do STPGS_UndFInv[I] := dm_ICM.t_SS7_STP.FieldByName('CLinks').AsFloat / 2.0 *
1254  Clink_UndFInv[I];
1255        for I := 1 to 2 do STPGS_PoleInv[I] := dm_ICM.t_SS7_STP.FieldByName('CLinks').AsFloat / 2.0 *
1256  Clink_PoleInv[I];
1257        for I := 1 to 2 do STPGS_CondInv[I] := dm_ICM.t_SS7_STP.FieldByName('CLinks').AsFloat / 2.0 *
1258  Clink_CondInv[I];
1259    //RFPC0018 Section VI Part 5 end
1260
1261    //RFPC0018 Section VI Part 6 begin
1262    {
1263          6. Port Units for STP Getting Started Equipment
1264
1265             Number of ports is required for unitizing the STP Getting Started
1266             investments.
1267
1268             Use value for STPMtceGrowth, the STP Maintenance/Growth Adjustment
1269             factor, and STPFill, the STP objective fill factor for ports, from
1270             the SS7 Network Parameters Table.
1271
1272          Average Investments:
1273
1274             STPGS_Ports = RoundUp (STP_Table.Ports_InSvc * (1 + STPMtceGrowth)),
1275             where STP_Table.Ports_InSvc = Number of ports in service for this
1276             STP_CLLI.
1277
1278          Volume-Sensitive Investments:
1279
1280             STPGS_Ports = RoundDown (STP_Table.PortCap * STPFill)
1281             where STP_Table.PortCap = STP Port Capacity for this STP_CLLI.
1282    }
1283        X := GetParameter('STPMtceGrowth');
```

```
1284        STPGS_Ports[1] := Ceil(dm_ICM.t_SS7_STP.FieldByName('Ports_InSvc').AsFloat * (1.0+X));
1285
1286        X := GetParameter('STPFill');
1287        STPGS_Ports[2] := Int(dm_ICM.t_SS7_STP.FieldByName('PortCap').AsFloat * X);
1288        if STPGS_Ports[1] > STPGS_Ports[2] then STPGS_Ports[2] := STPGS_Ports[1];
1289    //RFPC0018 Section VI Part 6 end
1290
1291    //RFPC0018 Section VI Part 7 begin
1292    {
1293       7.  Investment per STP Port, by account
1294
1295       STPPORTSWT = STPPortInv + (STPGS_SwtInv  / STPGS_Ports)
1296       STPPORTCKT = STPGS_CktInv / STPGS_Ports
1297       STPPORTAERF = STPGS_AerFInv / STPGS_Ports
1298       STPPORTBURF = STPGS_BurFInv / STPGS_Ports
1299       STPPORTUNDF = STPGS_UndFInv / STPGS_Ports
1300       STPPORTPOLE = STPGS_PoleInv / STPGS_Ports
1301       STPPORTCOND = STPGS_CondInv / STPGS_Ports
1302       STPPORTEXP = STPGSExp / STPGS_Ports        (Monthly value)
1303    }
1304        for I := 1 to 2 do
1305          if STPGS_Ports[I] > 0.0
1306            then begin
1307                   STPPORTSWT[I]  := STPPortInv[I] + STPGS_SwtInv[I] / STPGS_Ports[I];
1308                   STPPORTCKT[I]  := STPGS_CktInv[I]  / STPGS_Ports[I];
1309                   STPPORTAERF[I] := STPGS_AerFInv[I] / STPGS_Ports[I];
1310                   STPPORTBURF[I] := STPGS_BurFInv[I] / STPGS_Ports[I];
1311                   STPPORTUNDF[I] := STPGS_UndFInv[I] / STPGS_Ports[I];
1312                   STPPORTPOLE[I] := STPGS_PoleInv[I] / STPGS_Ports[I];
1313                   STPPORTCOND[I] := STPGS_CondInv[I] / STPGS_Ports[I];
1314                   STPPORTEXP[I]  := STPGSExp / STPGS_Ports[I];
1315                 end
1316            else begin
1317                   STPPORTSWT[I]  := 0.0;
1318                   STPPORTCKT[I]  := 0.0;
1319                   STPPORTAERF[I] := 0.0;
1320                   STPPORTBURF[I] := 0.0;
1321                   STPPORTUNDF[I] := 0.0;
1322                   STPPORTPOLE[I] := 0.0;
```

```
1323                    STPPORTCOND[I] := 0.0;
1324                    STPPORTEXP[I]  := 0.0;
1325                  end;
1326    //RFPC0018 Section VI Part 7 end
1327
1328    //RFPC0018 Section VI Part 8 begin
1329    {
1330         8.  GTT Software Investment per TCAP Message
1331
1332             Global Title Translation (GTT) software is required for TCAP messages.
1333             Value is from the SS7 Investment Table, and the capacity per port is
1334             based upon the number of ports from the STP Table and the estimated
1335             message capacity per port from the Network Parameters Table.
1336
1337             Total GTT Investment at Local STP:
1338
1339             GTTInv_L  = Total Investment value "STP_GTTSW" for
1340             STP_Table.Switch_Type = STP_Table.STP_Type for STP_CLLI
1341             where SS7_Invst.Item = "STP_GTTSW"
1342
1343             Estimated Message Capacity and Demand per Port:
1344
1345             GTTMsgCap = Value from Network Parameters Table
1346             GTTMsgDem = ALinkUtil * GTTMsgCap,
1347             where ALinkUtil is value from Network Parameters Table.
1348
1349             GTT Average and volume-sensitive per TCAP Message at Local STP:
1350
1351             TCAPSWL = GTTInv_L / (GTTMsgDem * STP_Table.Ports_InSvc) for average
1352             investments and
1353             TCAPSWL = GTTInv_L / (GTTMsgCap * STP_Table.Ports_InSvc) for
1354             volume-sensitive investments,
1355             where STP_Table.Ports_InSvc = Number of ports in service for this
1356             STP_CLLI
1357    }
1358        if GetSS7Temp(CLLI, 'STP_GTTSW')
1359          then begin
1360                 GTTInv_L[1] := t_T1_SS7.FieldByName('Total Inv').AsFloat;
1361                 GTTInv_L[2] := t_T1_SS7.FieldByName('VS Inv').AsFloat;
```

```
1362              end;
1363
1364        GTTMsgCap := GetParameter('GTTMsgCap');
1365        GTTMsgDem := GTTMsgCap * GetParameter('ALinkUtil');
1366
1367        if GTTMsgDem > 0.0
1368           then TCAPSWL[1] := GTTInv_L[1] / (GTTMsgDem * dm_ICM.t_SS7_STP.FieldByName('Ports_InSvc').AsFloat)
1369           else TCAPSWL[1] := 0.0;
1370        if GTTMsgCap > 0.0
1371           then TCAPSWL[2] := GTTInv_L[2] / (GTTMsgCap * dm_ICM.t_SS7_STP.FieldByName('Ports_InSvc').AsFloat)
1372           else TCAPSWL[2] := 0.0;
1373     //RFPC0018 Section VI Part 8 end
1374
1375       //RFPC0018 write data out to a temporary table
1376         WriteTemp(CLLI, 'STPPortInv',      STPPortInv);
1377         WriteTemp(CLLI, 'STPSpec_CktInv',  STPSpec_CktInv);
1378         WriteTemp(CLLI, 'DS01Ratio',       DS01Ratio);
1379         WriteTemp(CLLI, 'DS13Ratio',       DS13Ratio);
1380         WriteTemp(CLLI, 'DSX3DS0',         DSX3DS0);
1381         WriteTemp(CLLI, 'DSX1DS0',         DSX1DS0);
1382         WriteTemp(CLLI, 'DCS10DS0',        DCS10DS0);
1383         WriteTemp(CLLI, 'Clink_CktInv',    Clink_CktInv);
1384         WriteTemp(CLLI, 'Clink_AerFInv',   CLink_AerFInv);
1385         WriteTemp(CLLI, 'Clink_BurFInv',   CLink_BurFInv);
1386         WriteTemp(CLLI, 'Clink_UndFInv',   CLink_UndFInv);
1387         WriteTemp(CLLI, 'Clink_PoleInv',   CLink_PoleInv);
1388         WriteTemp(CLLI, 'Clink_CondInv',   CLink_CondInv);
1389         WriteTemp(CLLI, 'STPGS_SwtInv',    STPGS_SwtInv);
1390         WriteTemp(CLLI, 'STPGS_CktInv',    STPGS_CktInv);
1391         WriteTemp(CLLI, 'STPGS_AerFInv',   STPGS_AerFInv);
1392         WriteTemp(CLLI, 'STPGS_BurFInv',   STPGS_BurFInv);
1393         WriteTemp(CLLI, 'STPGS_UndFInv',   STPGS_UndFInv);
1394         WriteTemp(CLLI, 'STPGS_PoleInv',   STPGS_PoleInv);
1395         WriteTemp(CLLI, 'STPGS_CondInv',   STPGS_CondInv);
1396         WriteTemp(CLLI, 'STPGS_Ports',     STPGS_Ports);
1397         WriteTemp(CLLI, 'STPPORTSWT',      STPPORTSWT);
1398         WriteTemp(CLLI, 'STPPORTCKT',      STPPORTCKT);
1399         WriteTemp(CLLI, 'STPPORTAERF',     STPPORTAERF);
1400         WriteTemp(CLLI, 'STPPORTBURF',     STPPORTBURF);
```

```
1401        WriteTemp(CLLI, 'STPPORTUNDF',      STPPORTUNDF);
1402        WriteTemp(CLLI, 'STPPORTPOLE',      STPPORTPOLE);
1403        WriteTemp(CLLI, 'STPPORTCOND',      STPPORTCOND);
1404        WriteTemp(CLLI, 'STPPORTEXP',       STPPORTEXP);
1405        WriteTemp(CLLI, 'GTTInv_L',         GTTInv_L);
1406        WriteTemp(CLLI, 'TCAPSWL',          TCAPSWL);
1407        dm_ICM.t_SS7_STP.Next;
1408      end;
1409
1410    //FRPC0018 Section VII Preliminary Regional STP Calculations
1411    {
1412         VII.  Preliminary Regional STP Calculations, by STP (All STPs)
1413
1414               Unless otherwise noted, the calculations that follow apply uniformly
1415               to volume-sensitive and average investments.  The volume-insensitive
1416               investment is the difference between the average and volume-sensitive
1417               investments.
1418
1419               For a given STP_CLLI, let RSTP1 = value in the STP Table in the
1420               Regional_STP1 column, and RSTP2 = value in the STP Table in the
1421               Regional_STP2 column.
1422
1423               Investment per D Link to Regional STPs:
1424
1425               If STP_CLLI = RSTP1 or STP_CLLI = RSTP2, then set all investment
1426               and expense components of the Cost per D Link to 0 because the
1427               Local STP is also the Regional STP in this case.
1428    }
1429    dm_ICM.t_SS7_STP.First;           //RFPC0018 Go to the beginning of the STP table
1430    while Not dm_ICM.t_SS7_STP.EOF do
1431      begin
1432        Application.ProcessMessages;
1433        CLLI         := dm_ICM.t_SS7_STP.FieldByName('STP_CLLI').AsString;     //RFPC0018 Get Clli Code
1434        State        := Copy(CLLI, 5, 2);                                      //RFPC0018 Extract state code
1435        WireCtr_CLLI := dm_ICM.t_SS7_STP.FieldByName('WCenter_CLLI').AsString; //RFPC0018 Get equipment type
1436        GetInterOfficeValues;                                                        //RFPC0018 Get inter-
1437  office values
1438
1439        //RFPC0018 set values to zero
```

```
1440        for I := 1 to 3 do
1441          begin
1442            DLINKCKT[I]      := 0.0;
1443            DLINKAERF[I]     := 0.0;
1444            DLINKBURF[I]     := 0.0;
1445            DLINKUNDF[I]     := 0.0;
1446            DLINKPOLE[I]     := 0.0;
1447            DLINKCOND[I]     := 0.0;
1448            DLINKEXP[I]      := 0.0;
1449            DB800LINKCKT[I] := 0.0;
1450            LIDBLINKCKT[I]  := 0.0;
1451            ALINK800CKT[I]  := 0.0;
1452            ALINK800AERF[I] := 0.0;
1453            ALINK800BURF[I] := 0.0;
1454            ALINK800UNDF[I] := 0.0;
1455            ALINK800POLE[I] := 0.0;
1456            ALINK800COND[I] := 0.0;
1457            ALINKLIDBCKT[I] := 0.0;
1458            ALINKLIDBAERF[I]:= 0.0;
1459            ALINKLIDBBURF[I]:= 0.0;
1460            ALINKLIDBUNDF[I]:= 0.0;
1461            ALINKLIDBPOLE[I]:= 0.0;
1462            ALINKLIDBCOND[I]:= 0.0;
1463            RSPORTSWT[I]     := 0.0;
1464            RSPORTCKT[I]     := 0.0;
1465            RSPORTAERF[I]    := 0.0;
1466            RSPORTBURF[I]    := 0.0;
1467            RSPORTUNDF[I]    := 0.0;
1468            RSPORTPOLE[I]    := 0.0;
1469            RSPORTCOND[I]    := 0.0;
1470            GTTInv_R[I]      := 0.0;
1471            TCAPSWR[I]       := 0.0;
1472            DB800PORTSWT[I] := 0.0;
1473            DB800PORTCKT[I] := 0.0;
1474            DB800PORTAERF[I]:= 0.0;
1475            DB800PORTBURF[I]:= 0.0;
1476            DB800PORTUNDF[I]:= 0.0;
1477            DB800PORTPOLE[I]:= 0.0;
1478            DB800PORTCOND[I]:= 0.0;
```

```
1479            LIDBPORTSWT[I]  := 0.0;
1480            LIDBPORTCKT[I]  := 0.0;
1481            LIDBPORTAERF[I] := 0.0;
1482            LIDBPORTBURF[I] := 0.0;
1483            LIDBPORTUNDF[I] := 0.0;
1484            LIDBPORTPOLE[I] := 0.0;
1485            LIDBPORTCOND[I] := 0.0;
1486          end;
1487
1488        //RFPC0018 Check for a regional STP
1489        if (CLLI <> dm_ICM.t_SS7_STP.FieldByName('Regional_STP1').AsString) and
1490           (CLLI <> dm_ICM.t_SS7_STP.FieldByName('Regional_STP2').AsString)
1491          then begin
1492                 //RFPC0018 Section VII part 1 begin
1493    {
1494           1.   Investment per D Link to Regional STPs
1495
1496                Investment per D Link applies to the Local STPs that are in the
1497                same LATA as their Regional STPs.  The circuit equipment investment
1498                is derived from the results previously computed for C links.  The
1499                outside plant investments depend upon whether the links are on GTE
1500                facilities or not.
1501
1502                Circuit Equipment Investment:
1503
1504
1505                DLINKCKT = Average [CLink_CktInv for the Local STP + CLink_CktInv
1506                                    for RSTP1, CLink_CktInv for the Local STP +
1507                                    CLink_CktInv for RSTP2]
1508    }
1509                 //RFPC0018 D link circuit equipment investment
1510                 if GetSS7Temp( CLLI, 'Clink_CktInv')
1511                   then begin
1512                          DLINKCKT[1] := t_T1_SS7.FieldByName('Total Inv').AsFloat * 2.0;
1513                          DLINKCKT[2] := t_T1_SS7.FieldByName('VS Inv').AsFloat * 2.0;
1514                        end;
1515                 if GetSS7Temp( dm_ICM.t_SS7_STP.FieldByName('Regional_STP1').AsString, 'Clink_CktInv')
1516                   then begin
1517                          DLINKCKT[1] := DLINKCKT[1] + t_T1_SS7.FieldByName('Total Inv').AsFloat;
```

```
1518                        DLINKCKT[2] := DLINKCKT[2] + t_T1_SS7.FieldByName('VS Inv').AsFloat;
1519                      end;
1520                if GetSS7Temp( dm_ICM.t_SS7_STP.FieldByName('Regional_STP2').AsString, 'Clink_CktInv')
1521                  then begin
1522                        DLINKCKT[1] := DLINKCKT[1] + t_T1_SS7.FieldByName('Total Inv').AsFloat;
1523                        DLINKCKT[2] := DLINKCKT[2] + t_T1_SS7.FieldByName('VS Inv').AsFloat;
1524                      end;
1525                DLINKCKT[1] := DLINKCKT[1] / 2.0;
1526                DLINKCKT[2] := DLINKCKT[2] / 2.0;
1527
1528                //RFPC0018 D link cable investment
1529                if dm_ICM.t_SS7_STP.FieldByName('DLink_Expense').AsFloat > 0.0
1530                  then begin
1531                        //RFPC0018 section VII Part 2
1532    {
1533            2.  Expense per D Link
1534
1535                DLINKEXP = STP_Table.DLink_Expense for the local STP being studied.
1536    }
1537                        DLINKEXP[1] := dm_ICM.t_SS7_STP.FieldByName('DLink_Expense').AsFloat;
1538                        DLINKEXP[2] := DLINKEXP[1];
1539                      end
1540                  else begin
1541    {
1542            Outside Plant Investments:
1543
1544            Following are the outside plant components.
1545
1546            DLINKAERF
1547            DLINKBURF
1548            DLINKUNDF
1549            DLINKPOLE
1550            DLINKCOND
1551
1552            If the local and regional STPs are not in the same LATA, then set
1553            these results to 0.  Otherwise, proceed with the following calculations.
1554            Use the transport investments and unit values from the Acth2h table
1555            where CLLI Code = SS7_Invst.WCenter_CLLI for the local STP.
1556
```

```
1557
1558            Latitude and Longitude of Regional STPs
1559
1560            Lat_RSTP1 = STP_Table.Latitude where STP_Table.STP_CLLI = RSTP1
1561            Long_RSTP1 = STP_Table.Longitude where STP_Table.STP_CLLI = RSTP1
1562
1563            Lat_RSTP2 = STP_Table.Latitude where STP_Table.STP_CLLI = RSTP2
1564            Long_RSTP2 = STP_Table.Longitude where STP_Table.STP_CLLI = RSTP2
1565
1566          Distances to Regional STPs
1567
1568            Distance should be calculated for each of the D links between the STP
1569            and its regional STPs:
1570
1571            Let ALMRATIO, the route mile to airline mile ratio.
1572
1573          STP to Regional STP 1:
1574
1575            D1Distance = ALMRATIO * Distance(Latitude, Longitude, Lat_RSTP1,
1576                         Long_RSTP1)
1577            Where Latitude and Longitude are the coordinates of the given STP_CLLI.
1578
1579          STP to Regional STP 2:
1580
1581            D2Distance = ALMRATIO * Distance(Latitude, Longitude, Lat_RSTP2,
1582                        Long_RSTP2)
1583            Where Latitude and Longitude are the coordinates of the given STP_CLLI.
1584
1585        Average Distance of D Links:
1586
1587             DDistance  = (D1Distance + D2Distance)/2
1588
1589        DS0/DS1 and DS1/DS3 Ratios
1590
1591             DS01Ratio = (Round Up (IHCBDS0/24) ) / IHDSX1DS1
1592             DS13Ratio = (Round Up (IHDSX1DS1/28) ) / IHDSX3DS3
1593             where IHCBDS0, IHDSX1DS1, and IHDSX3DS3 are values from table "Acth2h"
1594
1595
```

```
1596              Let TOTALDIST = IHAERDIST + IHBURDIST + IHUNGDIST,
1597              where IHAERDIST, IHBURDIST, and IHUNGDIST are the route distance
1598              values for aerial, buried, and underground cable, respectively,
1599              from the table "Acth2h"
1600
1601              DLINKAERF = (IHAERFI  / DSODISTANCE) * DDistance
1602
1603              DLINKBURF = (IHBURFI  / DSODISTANCE) * DDistance
1604
1605              DLINKUNDF = (IHUNDFI / DSODISTANCE) * DDistance
1606
1607              DLINK_POLE = (IHPOLES / DSODISTANCE) * DDistance
1608
1609              DLINK_COND = (IHCOND / DSODISTANCE) * DDistance
1610    }
1611                          X := GetParameter('ALMRATIO');
1612                          Distance := (GetDistance(CLLI,
1613  dm_ICM.t_SS7_STP.FieldByName('Regional_STP1').AsString) +
1614                                       GetDistance(CLLI,
1615  dm_ICM.t_SS7_STP.FieldByName('Regional_STP2').AsString));
1616                          Distance := Distance * X / 2.0;
1617
1618                           for I := 1 to 2 do
1619                           if DSODistance[I] > 0.0
1620                             then begin
1621                                  //RFPC0018 Calculate the DLink's aerial cable invesment
1622                                   DlinkAerF[I] := IHAERFI[I] / DSODistance[I] * Distance;
1623                                   //RFPC0018 Calculate the DLink's buried cable invesment
1624                                   DlinkBurF[I] := IHBURFI[I] / DSODistance[I] * Distance;
1625                                   //RFPC0018 Calculate the DLink's underground cable invesment
1626                                   DlinkUndF[I] := IHUNDFI[I] / DSODistance[I] * Distance;
1627                                   //RFPC0018 Calculate the DLink's poles invesment
1628                                   DlinkPole[I] := IHPOLES[I] / DSODistance[I] * Distance;
1629                                   //RFPC0018 Calculate the DLink's conduit invesment
1630                                   DlinkCond[I] := IHCOND[I] / DSODistance[I] * Distance;
1631                                 end;
1632                        end;
1633               //RFPC0018 Section VII part 1 end
1634
```

```
1635                end;
1636
1637          //RFPC0018 Section VII part 3 begin
1638      {
1639          3.  Investment and Expense per D Link to 800 Database STPs
1640
1641              These equations use the Regional STP Location Table items
1642              DB800_PriSCP1, DB800_PriSCP2, DB800_SecSCP1, and DB800_SecSCP2,
1643              which are the CLLI codes for the primary and secondary pairs of
1644              800 database SCPs, respectively.
1645
1646              If RSTP1 not in [DB800_PriSCP1, DB800_PriSCP2, DB800_SecSCP1, DB800_SecSCP2]:
1647
1648              800LINKCKT = Average(CLink_CktInv for RSTP1 + CLink_CktInv for
1649                           DB800_PriSCP1, CLink_CktInv for RSTP1 + CLink_CktInv for
1650                           DB800_PriSCP2, CLink_CktInv for RSTP1 + CLink_CktInv for
1651                           DB800_SecSCP1, CLink_CktInv for RSTP1 + CLink_CktInv for
1652                           DB800_SecSCP2, CLink_CktInv for RSTP2 + CLink_CktInv for
1653                           DB800_PriSCP1, CLink_CktInv for RSTP2 + CLink_CktInv for
1654                           DB800_PriSCP2, CLink_CktInv for RSTP2 + CLink_CktInv for
1655                           DB800_SecSCP1, CLink_CktInv for RSTP2 + CLink_CktInv for
1656                           DB800_SecSCP2)
1657
1658              If RSTP1 in [DB800_PriSCP1, DB800_PriSCP2]:
1659
1660              800LINKCKT = Average(CLink_CktInv for RSTP1 + CLink_CktInv for
1661                           DB800_SecSCP1, CLink_CktInv for RSTP1 + CLink_CktInv for
1662                           DB800_SecSCP2, CLink_CktInv for RSTP2+ CLink_CktInv for
1663                           DB800_SecSCP1, CLink_CktInv for RSTP2+ CLink_CktInv for
1664                           DB800_SecSCP2)
1665
1666              If RSTP1 in [DB800_SecSCP1, DB800_SecSCP2]:
1667
1668              800LINKCKT = Average(CLink_CktInv for RSTP1 + CLink_CktInv for
1669                          DB800_PriSCP1, CLink_CktInv for RSTP1 + CLink_CktInv for
1670                          DB800_PriSCP2, CLink_CktInv for RSTP2 + CLink_CktInv for
1671                          DB800_PriSCP1, CLink_CktInv for RSTP2 + CLink_CktInv for
1672                          DB800_PriSCP2)
1673
```

```
1674            800LINKEXP = STP_Table.DB800Link_Expense for the local STP being studied.
1675    }
1676        RCLLI := dm_ICM.t_SS7_STP.FieldByName('Regional_STP1').AsString;
1677        DB800LINKEXP[1] := dm_ICM.t_SS7_STP.FieldByName('DB800Link_Exp').AsFloat;
1678        DB800LINKEXP[2] := DB800LINKEXP[1];
1679
1680        if dm_ICM.t_SS7_Location.FindKey([RCLLI])
1681          then begin
1682                 //RFPC0018 if it's not one of these [DB800_PriSCP1, DB800_PriSCP2, DB800_SecSCP1,
1683    DB800_SecSCP2]
1684                 if (RCLLI <> dm_ICM.t_SS7_Location.FieldByName('DB800_PriSCP1').AsString) and
1685                    (RCLLI <> dm_ICM.t_SS7_Location.FieldByName('DB800_PriSCP2').AsString) and
1686                    (RCLLI <> dm_ICM.t_SS7_Location.FieldByName('DB800_SecSCP1').AsString) and
1687                    (RCLLI <> dm_ICM.t_SS7_Location.FieldByName('DB800_SecSCP2').AsString)
1688                   then begin
1689                          if GetSS7Temp( RCLLI, 'Clink_CktInv')
1690                            then begin
1691                                   DB800LINKCKT[1]:=t_T1_SS7.FieldByName('Total Inv').AsFloat * 4.0;
1692                                   DB800LINKCKT[2]:=t_T1_SS7.FieldByName('VS Inv').AsFloat * 4.0;
1693                                 end;
1694                          RCLLI := dm_ICM.t_SS7_STP.FieldByName('Regional_STP2').AsString;
1695                          if GetSS7Temp( RCLLI, 'Clink_CktInv')
1696                            then begin
1697                                   DB800LINKCKT[1]:= DB800LINKCKT[1]+t_T1_SS7.FieldByName('Total
1698    Inv').AsFloat*4.0;
1699                                   DB800LINKCKT[2]:= DB800LINKCKT[2]+t_T1_SS7.FieldByName('VS
1700    Inv').AsFloat*4.0;
1701                                 end;
1702                          RCLLI := dm_ICM.t_SS7_Location.FieldByName('DB800_PriSCP1').AsString;
1703                          if GetSS7Temp( RCLLI, 'Clink_CktInv')
1704                            then begin
1705                                   DB800LINKCKT[1]:= DB800LINKCKT[1]+t_T1_SS7.FieldByName('Total
1706    Inv').AsFloat*2.0;
1707                                   DB800LINKCKT[2]:= DB800LINKCKT[2]+t_T1_SS7.FieldByName('VS
1708    Inv').AsFloat*2.0;
1709                                 end;
1710                          RCLLI := dm_ICM.t_SS7_Location.FieldByName('DB800_PriSCP2').AsString;
1711                          if GetSS7Temp( RCLLI, 'Clink_CktInv')
1712                            then begin
```

```
1713                                    DB800LINKCKT[1]:= DB800LINKCKT[1]+t_T1_SS7.FieldByName('Total
1714  Inv').AsFloat*2.0;
1715                                    DB800LINKCKT[2]:= DB800LINKCKT[2]+t_T1_SS7.FieldByName('VS
1716  Inv').AsFloat*2.0;
1717                                 end;
1718                              RCLLI := dm_ICM.t_SS7_Location.FieldByName('DB800_SecSCP1').AsString;
1719                              if GetSS7Temp( RCLLI, 'Clink_CktInv')
1720                                then begin
1721                                    DB800LINKCKT[1]:= DB800LINKCKT[1]+t_T1_SS7.FieldByName('Total
1722  Inv').AsFloat*2.0;
1723                                    DB800LINKCKT[2]:= DB800LINKCKT[2]+t_T1_SS7.FieldByName('VS
1724  Inv').AsFloat*2.0;
1725                                 end;
1726                              RCLLI := dm_ICM.t_SS7_Location.FieldByName('DB800_SecSCP2').AsString;
1727                              if GetSS7Temp( RCLLI, 'Clink_CktInv')
1728                                then begin
1729                                    DB800LINKCKT[1]:= DB800LINKCKT[1]+t_T1_SS7.FieldByName('Total
1730  Inv').AsFloat*2.0;
1731                                    DB800LINKCKT[2]:= DB800LINKCKT[2]+t_T1_SS7.FieldByName('VS
1732  Inv').AsFloat*2.0;
1733                                 end;
1734                              DB800LINKCKT[1]:= DB800LINKCKT[1] / 8.0;
1735                              DB800LINKCKT[2]:= DB800LINKCKT[2] / 8.0;
1736                            end
1737
1738                     //RFPC0018 if it is one of these [DB800_PriSCP1, DB800_PriSCP2]
1739                      else if (RCLLI = dm_ICM.t_SS7_Location.FieldByName('DB800_PriSCP1').AsString) or
1740                              (RCLLI = dm_ICM.t_SS7_Location.FieldByName('DB800_PriSCP2').AsString)
1741                              then begin
1742                                    if GetSS7Temp( RCLLI, 'Clink_CktInv')
1743                                      then begin
1744                                             DB800LINKCKT[1]:=t_T1_SS7.FieldByName('Total Inv').AsFloat * 2.0;
1745                                             DB800LINKCKT[2]:=t_T1_SS7.FieldByName('VS Inv').AsFloat * 2.0;
1746                                           end;
1747                                    RCLLI := dm_ICM.t_SS7_STP.FieldByName('Regional_STP2').AsString;
1748                                    if GetSS7Temp( RCLLI, 'Clink_CktInv')
1749                                      then begin
1750                                             DB800LINKCKT[1]:= DB800LINKCKT[1] +
1751                                                               t_T1_SS7.FieldByName('Total Inv').AsFloat*2.0;
```

```
1752                                    DB800LINKCKT[2]:= DB800LINKCKT[2] +
1753                                                      t_T1_SS7.FieldByName('VS Inv').AsFloat*2.0;
1754                                  end;
1755                              RCLLI := dm_ICM.t_SS7_Location.FieldByName('DB800_SecSCP1').AsString;
1756                              if GetSS7Temp( RCLLI, 'Clink_CktInv')
1757                                then begin
1758                                    DB800LINKCKT[1]:= DB800LINKCKT[1] +
1759                                                      t_T1_SS7.FieldByName('Total Inv').AsFloat*2.0;
1760                                    DB800LINKCKT[2]:= DB800LINKCKT[2] +
1761                                                      t_T1_SS7.FieldByName('VS Inv').AsFloat*2.0;
1762                                  end;
1763                              RCLLI := dm_ICM.t_SS7_Location.FieldByName('DB800_SecSCP2').AsString;
1764                              if GetSS7Temp( RCLLI, 'Clink_CktInv')
1765                                then begin
1766                                    DB800LINKCKT[1]:= DB800LINKCKT[1] +
1767                                                      t_T1_SS7.FieldByName('Total Inv').AsFloat*2.0;
1768                                    DB800LINKCKT[2]:= DB800LINKCKT[2] +
1769                                                      t_T1_SS7.FieldByName('VS Inv').AsFloat*2.0;
1770                                  end;
1771                              DB800LINKCKT[1]:= DB800LINKCKT[1] / 4.0;
1772                              DB800LINKCKT[2]:= DB800LINKCKT[2] / 4.0;
1773                            end
1774
1775                        //RFPC0018 if it is one of these [DB800_SecSCP1, DB800_SecSCP2]
1776                        else if (RCLLI = dm_ICM.t_SS7_Location.FieldByName('DB800_SecSCP1').AsString) or
1777                                (RCLLI = dm_ICM.t_SS7_Location.FieldByName('DB800_SecSCP2').AsString)
1778                              then begin
1779                                    if GetSS7Temp( RCLLI, 'Clink_CktInv')
1780                                      then begin
1781                                            DB800LINKCKT[1]:=t_T1_SS7.FieldByName('Total Inv').AsFloat *
1782  2.0;
1783                                            DB800LINKCKT[2]:=t_T1_SS7.FieldByName('VS Inv').AsFloat *
1784  2.0;
1785                                          end;
1786                                    RCLLI := dm_ICM.t_SS7_STP.FieldByName('Regional_STP2').AsString;
1787                                    if GetSS7Temp( RCLLI, 'Clink_CktInv')
1788                                      then begin
1789                                            DB800LINKCKT[1]:= DB800LINKCKT[1] +
```

```
1790                                                                    t_T1_SS7.FieldByName('Total
1791    Inv').AsFloat*2.0;
1792                                                                  DB800LINKCKT[2]:= DB800LINKCKT[2] +
1793                                                                    t_T1_SS7.FieldByName('VS
1794    Inv').AsFloat*2.0;
1795                                                          end;
1796                                                        RCLLI := dm_ICM.t_SS7_Location.FieldByName('DB800_PriSCP1').AsString;
1797                                                        if GetSS7Temp( RCLLI, 'Clink_CktInv')
1798                                                          then begin
1799                                                            DB800LINKCKT[1]:= DB800LINKCKT[1] +
1800                                                              t_T1_SS7.FieldByName('Total
1801    Inv').AsFloat*2.0;
1802                                                            DB800LINKCKT[2]:= DB800LINKCKT[2] +
1803                                                              t_T1_SS7.FieldByName('VS
1804    Inv').AsFloat*2.0;
1805                                                          end;
1806                                                        RCLLI := dm_ICM.t_SS7_Location.FieldByName('DB800_PriSCP2').AsString;
1807                                                        if GetSS7Temp( RCLLI, 'Clink_CktInv')
1808                                                          then begin
1809                                                            DB800LINKCKT[1]:= DB800LINKCKT[1] +
1810                                                              t_T1_SS7.FieldByName('Total
1811    Inv').AsFloat*2.0;
1812                                                            DB800LINKCKT[2]:= DB800LINKCKT[2] +
1813                                                              t_T1_SS7.FieldByName('VS
1814    Inv').AsFloat*2.0;
1815                                                          end;
1816                                                        DB800LINKCKT[1]:= DB800LINKCKT[1] / 4.0;
1817                                                        DB800LINKCKT[2]:= DB800LINKCKT[2] / 4.0;
1818                                                      end;
1819                  end;
1820          //RFPC0018 Section VII part 3 end;
1821
1822          //RFPC0018 Section VII part 4 begin
1823     {
1824          4.  Investment and Expense per D Link to LIDB Database STPs
1825
1826              These equations use the Regional STP Location Table items LIDB_PriSCP1,
1827              LIDB_PriSCP2, LIDB_SecSCP1, and LIDB_SecSCP2, which are the CLLI codes
1828              for the primary and secondary pairs of LIDB database SCPs, respectively.
```

```
1829
1830            If RSTP1 not in [LIDB_PriSCP1, LIDB_PriSCP2, LIDB_SecSCP1, LIDB_SecSCP2]:
1831
1832            LIDBLINKCKT = Average(CLink_CktInv for RSTP1 + CLink_CktInv for
1833                          LIDB_PriSCP1, CLink_CktInv for RSTP1 + CLink_CktInv for
1834                          LIDB_PriSCP2, CLink_CktInv for RSTP1 + CLink_CktInv for
1835                          LIDB_SecSCP1, CLink_CktInv for RSTP1 + CLink_CktInv for
1836                          LIDB_SecSCP2, CLink_CktInv for RSTP2 + CLink_CktInv for
1837                          LIDB_PriSCP1, CLink_CktInv for RSTP2 + CLink_CktInv for
1838                          LIDB_PriSCP2, CLink_CktInv for RSTP2 + CLink_CktInv for
1839                          LIDB_SecSCP1, CLink_CktInv for RSTP2 + CLink_CktInv for
1840                          LIDB_SecSCP2)
1841
1842            If RSTP1 in [LIDB_PriSCP1, LIDB_PriSCP2]:
1843
1844            LIDBLINKCKT = Average(CLink_CktInv for RSTP1 + CLink_CktInv for
1845                          LIDB_SecSCP1, CLink_CktInv for RSTP1 + CLink_CktInv for
1846                          LIDB_SecSCP2, CLink_CktInv for RSTP2+ CLink_CktInv for
1847                          LIDB_SecSCP1, CLink_CktInv for RSTP2+ CLink_CktInv for
1848                          LIDB_SecSCP2)
1849
1850            If RSTP1 in [LIDB_SecSCP1, LIDB_SecSCP2]:
1851
1852            LIDBLINKCKT = Average(CLink_CktInv for RSTP1 + CLink_CktInv for
1853                          LIDB_PriSCP1, CLink_CktInv for RSTP1 + CLink_CktInv for
1854                          LIDB_PriSCP2, CLink_CktInv for RSTP2 + CLink_CktInv for
1855                          LIDB_PriSCP1, CLink_CktInv for RSTP2 + CLink_CktInv for
1856                          LIDB_PriSCP2)
1857
1858
1859             LIDBLINKEXP = STP_Table.LIDBLink_Expense for the local STP being studied.
1860    }
1861        RCLLI := dm_ICM.t_SS7_STP.FieldByName('Regional_STP1').AsString;
1862        LIDBLINKEXP[1] := dm_ICM.t_SS7_STP.FieldByName('LIDBLink_Exp').AsFloat;
1863        LIDBLINKEXP[2] := LIDBLINKEXP[1];
1864
1865        if dm_ICM.t_SS7_Location.FindKey([RCLLI])
1866          then begin
1867                //RFPC0018 if it's not one of these [LIDB_PriSCP1, LIDB_PriSCP2, LIDB_SecSCP1, LIDB_SecSCP2]
```

```
1868                  if (RCLLI <> dm_ICM.t_SS7_Location.FieldByName('LIDB_PriSCP1').AsString) and
1869                     (RCLLI <> dm_ICM.t_SS7_Location.FieldByName('LIDB_PriSCP2').AsString) and
1870                     (RCLLI <> dm_ICM.t_SS7_Location.FieldByName('LIDB_SecSCP1').AsString) and
1871                     (RCLLI <> dm_ICM.t_SS7_Location.FieldByName('LIDB_SecSCP2').AsString)
1872                    then begin
1873                          if GetSS7Temp( RCLLI, 'Clink_CktInv')
1874                            then begin
1875                                  LIDBLINKCKT[1]:=t_T1_SS7.FieldByName('Total Inv').AsFloat * 4.0;
1876                                  LIDBLINKCKT[2]:=t_T1_SS7.FieldByName('VS Inv').AsFloat * 4.0;
1877                                end;
1878                          RCLLI := dm_ICM.t_SS7_STP.FieldByName('Regional_STP2').AsString;
1879                          if GetSS7Temp( RCLLI, 'Clink_CktInv')
1880                            then begin
1881                                  LIDBLINKCKT[1]:= LIDBLINKCKT[1]+t_T1_SS7.FieldByName('Total
1882  Inv').AsFloat*4.0;
1883                                  LIDBLINKCKT[2]:= LIDBLINKCKT[2]+t_T1_SS7.FieldByName('VS
1884  Inv').AsFloat*4.0;
1885                                end;
1886                          RCLLI := dm_ICM.t_SS7_Location.FieldByName('LIDB_PriSCP1').AsString;
1887                          if GetSS7Temp( RCLLI, 'Clink_CktInv')
1888                            then begin
1889                                  LIDBLINKCKT[1]:= LIDBLINKCKT[1]+t_T1_SS7.FieldByName('Total
1890  Inv').AsFloat*2.0;
1891                                  LIDBLINKCKT[2]:= LIDBLINKCKT[2]+t_T1_SS7.FieldByName('VS
1892  Inv').AsFloat*2.0;
1893                                end;
1894                          RCLLI := dm_ICM.t_SS7_Location.FieldByName('LIDB_PriSCP2').AsString;
1895                          if GetSS7Temp( RCLLI, 'Clink_CktInv')
1896                            then begin
1897                                  LIDBLINKCKT[1]:= LIDBLINKCKT[1]+t_T1_SS7.FieldByName('Total
1898  Inv').AsFloat*2.0;
1899                                  LIDBLINKCKT[2]:= LIDBLINKCKT[2]+t_T1_SS7.FieldByName('VS
1900  Inv').AsFloat*2.0;
1901                                end;
1902                          RCLLI := dm_ICM.t_SS7_Location.FieldByName('LIDB_SecSCP1').AsString;
1903                          if GetSS7Temp( RCLLI, 'Clink_CktInv')
1904                            then begin
1905                                  LIDBLINKCKT[1]:= LIDBLINKCKT[1]+t_T1_SS7.FieldByName('Total
1906  Inv').AsFloat*2.0;
```

```
1907                                    LIDBLINKCKT[2]:= LIDBLINKCKT[2]+t_T1_SS7.FieldByName('VS
1908    Inv').AsFloat*2.0;
1909                                  end;
1910                              RCLLI := dm_ICM.t_SS7_Location.FieldByName('LIDB_SecSCP2').AsString;
1911                              if GetSS7Temp( RCLLI, 'Clink_CktInv')
1912                                then begin
1913                                    LIDBLINKCKT[1]:= LIDBLINKCKT[1]+t_T1_SS7.FieldByName('Total
1914    Inv').AsFloat*2.0;
1915                                    LIDBLINKCKT[2]:= LIDBLINKCKT[2]+t_T1_SS7.FieldByName('VS
1916    Inv').AsFloat*2.0;
1917                                  end;
1918                              LIDBLINKCKT[1]:= LIDBLINKCKT[1] / 8.0;
1919                              LIDBLINKCKT[2]:= LIDBLINKCKT[2] / 8.0;
1920                            end
1921
1922                      //RFPC0018 if it is one of these {LIDB_PriSCP1, LIDB_PriSCP2}
1923                       else if (RCLLI = dm_ICM.t_SS7_Location.FieldByName('LIDB_PriSCP1').AsString) or
1924                               (RCLLI = dm_ICM.t_SS7_Location.FieldByName('LIDB_PriSCP2').AsString)
1925                              then begin
1926                                    if GetSS7Temp( RCLLI, 'Clink_CktInv')
1927                                      then begin
1928                                            LIDBLINKCKT[1]:=t_T1_SS7.FieldByName('Total Inv').AsFloat * 2.0;
1929                                            LIDBLINKCKT[2]:=t_T1_SS7.FieldByName('VS Inv').AsFloat * 2.0;
1930                                          end;
1931                                    RCLLI := dm_ICM.t_SS7_STP.FieldByName('Regional_STP2').AsString;
1932                                    if GetSS7Temp( RCLLI, 'Clink_CktInv')
1933                                      then begin
1934                                            LIDBLINKCKT[1]:= LIDBLINKCKT[1] +
1935                                                             t_T1_SS7.FieldByName('Total Inv').AsFloat*2.0;
1936                                            LIDBLINKCKT[2]:= LIDBLINKCKT[2] +
1937                                                             t_T1_SS7.FieldByName('VS Inv').AsFloat*2.0;
1938                                          end;
1939                                    RCLLI := dm_ICM.t_SS7_Location.FieldByName('LIDB_SecSCP1').AsString;
1940                                    if GetSS7Temp( RCLLI, 'Clink_CktInv')
1941                                      then begin
1942                                            LIDBLINKCKT[1]:= LIDBLINKCKT[1] +
1943                                                             t_T1_SS7.FieldByName('Total Inv').AsFloat*2.0;
1944                                            LIDBLINKCKT[2]:= LIDBLINKCKT[2] +
1945                                                             t_T1_SS7.FieldByName('VS Inv').AsFloat*2.0;
```

```
1946                                    end;
1947                                  RCLLI := dm_ICM.t_SS7_Location.FieldByName('LIDB_SecSCP2').AsString;
1948                                  if GetSS7Temp( RCLLI, 'Clink_CktInv')
1949                                    then begin
1950                                           LIDBLINKCKT[1]:= LIDBLINKCKT[1] +
1951                                                            t_T1_SS7.FieldByName('Total Inv').AsFloat*2.0;
1952                                           LIDBLINKCKT[2]:= LIDBLINKCKT[2] +
1953                                                            t_T1_SS7.FieldByName('VS Inv').AsFloat*2.0;
1954                                    end;
1955                                  LIDBLINKCKT[1]:= LIDBLINKCKT[1] / 4.0;
1956                                  LIDBLINKCKT[2]:= LIDBLINKCKT[2] / 4.0;
1957                                end
1958
1959                           //RFPC0018 if it is one of these [LIDB_SecSCP1, LIDB_SecSCP2]
1960                           else if (RCLLI = dm_ICM.t_SS7_Location.FieldByName('LIDB_SecSCP1').AsString) or
1961                                   (RCLLI = dm_ICM.t_SS7_Location.FieldByName('LIDB_SecSCP2').AsString)
1962                                  then begin
1963                                         if GetSS7Temp( RCLLI, 'Clink_CktInv')
1964                                           then begin
1965                                                  LIDBLINKCKT[1]:=t_T1_SS7.FieldByName('Total Inv').AsFloat *
1966   2.0;
1967                                                  LIDBLINKCKT[2]:=t_T1_SS7.FieldByName('VS Inv').AsFloat *
1968   2.0;
1969                                                end;
1970                                         RCLLI := dm_ICM.t_SS7_STP.FieldByName('Regional_STP2').AsString;
1971                                         if GetSS7Temp( RCLLI, 'Clink_CktInv')
1972                                           then begin
1973                                                  LIDBLINKCKT[1]:= LIDBLINKCKT[1] +
1974                                                                   t_T1_SS7.FieldByName('Total
1975   Inv').AsFloat*2.0;
1976                                                  LIDBLINKCKT[2]:= LIDBLINKCKT[2] +
1977                                                                   t_T1_SS7.FieldByName('VS
1978   Inv').AsFloat*2.0;
1979                                                end;
1980                                         RCLLI := dm_ICM.t_SS7_Location.FieldByName('LIDB_PriSCP1').AsString;
1981                                         if GetSS7Temp( RCLLI, 'Clink_CktInv')
1982                                           then begin
1983                                                  LIDBLINKCKT[1]:= LIDBLINKCKT[1] +
```

```
                                                                    t_T1_SS7.FieldByName('Total
Inv').AsFloat*2.0;
                                                  LIDBLINKCKT[2]:= LIDBLINKCKT[2] +
                                                                    t_T1_SS7.FieldByName('VS
Inv').AsFloat*2.0;
                                                end;
                                              RCLLI := dm_ICM.t_SS7_Location.FieldByName('LIDB_PriSCP2').AsString;
                                              if GetSS7Temp( RCLLI, 'Clink_CktInv')
                                                then begin
                                                  LIDBLINKCKT[1]:= LIDBLINKCKT[1] +
                                                                    t_T1_SS7.FieldByName('Total
Inv').AsFloat*2.0;
                                                  LIDBLINKCKT[2]:= LIDBLINKCKT[2] +
                                                                    t_T1_SS7.FieldByName('VS
Inv').AsFloat*2.0;
                                                end;
                                              LIDBLINKCKT[1]:= LIDBLINKCKT[1] / 4.0;
                                              LIDBLINKCKT[2]:= LIDBLINKCKT[2] / 4.0;
                                            end;
              end;
      //RFPC0018 Section VII part 4 end;


      //RFPC0018 Section VII part 7 begin
  {
        7. Investments for A Links to SCPs

           The SCPs for a database service are linked to the correponding pair
           of regional STPs by A links.  The SCPs are colocated with the STPs,
           so for a given location the A link between STP and SCP is simply the
           special circuit termination equipment required at both ends.
           However, an A link is also required between each of the regional
           STPs and the SCP at the mated STP location.  Therefore, the facility
           and DS3/DS1/DS0 circuit investments per A link to SCPs is half of
           the investments required for the distance between the mated pair
           locations.

           Link investments were previously calculated and stored for the C
           links between the STPs.  These values can be used here rather than
```

```
2023            recomputing these link investments.  Since C links of all of the
2024            database regional STPs are on GTE facilities, no lease expenses are
2025            involved.  The C link investment values used here are for the
2026            database regional STPs serving this local STP through another
2027            regional STP.
2028
2029            In the equations below, C link investments are divided by 8 so that,
2030            at the SSP level, results for the two local STPs will be averaged,
2031            yielding half of the investment for one C link.  An adjustment to
2032            the circuit equipment investment adds back one-half of the STP
2033            special circuit investment so that the special circuit portion of
2034            the A link has the same value as the C links on the same route.
2035            When the special circuit investment portions are averaged between
2036            the two local STPs, investments for two full special circuit
2037            terminations result.
2038        }
2039          RCLLI := dm_ICM.t_SS7_STP.FieldByName('Regional_STP1').AsString;
2040          if CLLI = RCLLI
2041            then begin
2042                   if dm_ICM.t_SS7_Location.FindKey([RCLLI])
2043                     then begin
2044                            //RFPC0018 ----------------------------------------
2045                            //RFPC0018 A link for database 800 primary SCP1
2046                            RCLLI := dm_ICM.t_SS7_Location.FieldByName('DB800_PriSCP1').AsString;
2047
2048                            //RFPC0018 A link for database 800 circuit equipment
2049                            if GetSS7Temp( RCLLI, 'Clink_CktInv')
2050                             then begin
2051                                    ALINK800CKT[1] := ALINK800CKT[1] + t_T1_SS7.FieldByName('Total
2052  Inv').AsFloat;
2053                                    ALINK800CKT[2] := ALINK800CKT[2] + t_T1_SS7.FieldByName('VS Inv').AsFloat;
2054                                  end;
2055
2056                            //RFPC0018 A link for database 800 circuit equipment
2057                            if GetSS7Temp( RCLLI, 'STPSpec_CktInv')
2058                             then begin
2059                                    ALINK800CKT[1] := ALINK800CKT[1] + t_T1_SS7.FieldByName('Total
2060  Inv').AsFloat;
2061                                    ALINK800CKT[2] := ALINK800CKT[2] + t_T1_SS7.FieldByName('VS Inv').AsFloat;
```

```
2062                            end;
2063
2064                          //RFPC0018 A link for database 800 aerial cable
2065                          if GetSS7Temp( RCLLI, 'Clink_AerFInv')
2066                           then begin
2067                                  ALINK800AERF[1] := ALINK800AERF[1] + t_T1_SS7.FieldByName('Total
2068   Inv').AsFloat;
2069                                  ALINK800AERF[2] := ALINK800AERF[2] + t_T1_SS7.FieldByName('VS
2070   Inv').AsFloat;
2071                            end;
2072
2073                          //RFPC0018 A link for database 800 buried cable
2074                          if GetSS7Temp( RCLLI, 'Clink_BurFInv')
2075                           then begin
2076                                  ALINK800BURF[1] := ALINK800BURF[1] + t_T1_SS7.FieldByName('Total
2077   Inv').AsFloat;
2078                                  ALINK800BURF[2] := ALINK800BURF[2] + t_T1_SS7.FieldByName('VS
2079   Inv').AsFloat;
2080                            end;
2081
2082                          //RFPC0018 A link for database 800 underground cable
2083                          if GetSS7Temp( RCLLI, 'Clink_UndFInv')
2084                           then begin
2085                                  ALINK800UNDF[1] := ALINK800UNDF[1] + t_T1_SS7.FieldByName('Total
2086   Inv').AsFloat;
2087                                  ALINK800UNDF[2] := ALINK800UNDF[2] + t_T1_SS7.FieldByName('VS
2088   Inv').AsFloat;
2089                            end;
2090
2091                          //RFPC0018 A link for database 800 poles
2092                          if GetSS7Temp( RCLLI, 'Clink_PoleInv')
2093                           then begin
2094                                  ALINK800POLE[1] := ALINK800POLE[1] + t_T1_SS7.FieldByName('Total
2095   Inv').AsFloat;
2096                                  ALINK800POLE[2] := ALINK800POLE[2] + t_T1_SS7.FieldByName('VS
2097   Inv').AsFloat;
2098                            end;
2099
2100                          //RFPC0018 A link for database 800 conduit
```

```
2101                        if GetSS7Temp( RCLLI, 'Clink_CondInv')
2102                         then begin
2103                                ALINK800COND[1] := ALINK800COND[1] + t_T1_SS7.FieldByName('Total
2104  Inv').AsFloat;
2105                                ALINK800COND[2] := ALINK800COND[2] + t_T1_SS7.FieldByName('VS
2106  Inv').AsFloat;
2107                             end;
2108
2109                        //RFPC0018 ------------------------------------------
2110                        //RFPC0018 A link for database 800 primary SCP2
2111                        RCLLI := dm_ICM.t_SS7_Location.FieldByName('DB800_PriSCP2').AsString;
2112
2113                        //RFPC0018 A link for database 800 circuit equipment
2114                        if GetSS7Temp( RCLLI, 'Clink_CktInv')
2115                         then begin
2116                                ALINK800CKT[1] := ALINK800CKT[1] + t_T1_SS7.FieldByName('Total
2117  Inv').AsFloat;
2118                                ALINK800CKT[2] := ALINK800CKT[2] + t_T1_SS7.FieldByName('VS Inv').AsFloat;
2119                             end;
2120
2121                        //RFPC0018 A link for database 800 circuit equipment
2122                        if GetSS7Temp( RCLLI, 'STPSpec_CktInv')
2123                         then begin
2124                                ALINK800CKT[1] := ALINK800CKT[1] + t_T1_SS7.FieldByName('Total
2125  Inv').AsFloat;
2126                                ALINK800CKT[2] := ALINK800CKT[2] + t_T1_SS7.FieldByName('VS Inv').AsFloat;
2127                             end;
2128
2129                        //RFPC0018 A link for database 800 aerial cable
2130                        if GetSS7Temp( RCLLI, 'Clink_AerFInv')
2131                         then begin
2132                                ALINK800AERF[1] := ALINK800AERF[1] + t_T1_SS7.FieldByName('Total
2133  Inv').AsFloat;
2134                                ALINK800AERF[2] := ALINK800AERF[2] + t_T1_SS7.FieldByName('VS
2135  Inv').AsFloat;
2136                             end;
2137
2138                        //RFPC0018 A link for database 800 buried cable
2139                        if GetSS7Temp( RCLLI, 'Clink_BurFInv')
```

```
2140                          then begin
2141                                ALINK800BURF[1] := ALINK800BURF[1] + t_T1_SS7.FieldByName('Total
2142  Inv').AsFloat;
2143                                ALINK800BURF[2] := ALINK800BURF[2] + t_T1_SS7.FieldByName('VS
2144  Inv').AsFloat;
2145                           end;
2146
2147                         //RFPC0018 A link for database 800 underground cable
2148                         if GetSS7Temp( RCLLI, 'Clink_UndFInv')
2149                          then begin
2150                                ALINK800UNDF[1] := ALINK800UNDF[1] + t_T1_SS7.FieldByName('Total
2151  Inv').AsFloat;
2152                                ALINK800UNDF[2] := ALINK800UNDF[2] + t_T1_SS7.FieldByName('VS
2153  Inv').AsFloat;
2154                           end;
2155
2156                         //RFPC0018 A link for database 800 poles
2157                         if GetSS7Temp( RCLLI, 'Clink_PoleInv')
2158                          then begin
2159                                ALINK800POLE[1] := ALINK800POLE[1] + t_T1_SS7.FieldByName('Total
2160  Inv').AsFloat;
2161                                ALINK800POLE[2] := ALINK800POLE[2] + t_T1_SS7.FieldByName('VS
2162  Inv').AsFloat;
2163                           end;
2164
2165                         //RFPC0018 A link for database 800 conduit
2166                         if GetSS7Temp( RCLLI, 'Clink_CondInv')
2167                          then begin
2168                                ALINK800COND[1] := ALINK800COND[1] + t_T1_SS7.FieldByName('Total
2169  Inv').AsFloat;
2170                                ALINK800COND[2] := ALINK800COND[2] + t_T1_SS7.FieldByName('VS
2171  Inv').AsFloat;
2172                           end;
2173
2174                         //RFPC0018 ------------------------------------------
2175                         //RFPC0018 A link for database 800 secondary SCP1
2176                         RCLLI := dm_ICM.t_SS7_Location.FieldByName('DB800_SecSCP1').AsString;
2177
2178                         //RFPC0018 A link for database 800 circuit equipment
```

```
2179                        if GetSS7Temp( RCLLI, 'Clink_CktInv')
2180                         then begin
2181                                ALINK800CKT[1] := ALINK800CKT[1] + t_T1_SS7.FieldByName('Total
2182  Inv').AsFloat;
2183                                ALINK800CKT[2] := ALINK800CKT[2] + t_T1_SS7.FieldByName('VS Inv').AsFloat;
2184                              end;
2185
2186                        //RFPC0018 A link for database 800 circuit equipment
2187                        if GetSS7Temp( RCLLI, 'STPSpec_CktInv')
2188                         then begin
2189                                ALINK800CKT[1] := ALINK800CKT[1] + t_T1_SS7.FieldByName('Total
2190  Inv').AsFloat;
2191                                ALINK800CKT[2] := ALINK800CKT[2] + t_T1_SS7.FieldByName('VS Inv').AsFloat;
2192                              end;
2193
2194                        //RFPC0018 A link for database 800 aerial cable
2195                        if GetSS7Temp( RCLLI, 'Clink_AerFInv')
2196                         then begin
2197                                ALINK800AERF[1] := ALINK800AERF[1] + t_T1_SS7.FieldByName('Total
2198  Inv').AsFloat;
2199                                ALINK800AERF[2] := ALINK800AERF[2] + t_T1_SS7.FieldByName('VS
2200  Inv').AsFloat;
2201                              end;
2202
2203                        //RFPC0018 A link for database 800 buried cable
2204                        if GetSS7Temp( RCLLI, 'Clink_BurFInv')
2205                         then begin
2206                                ALINK800BURF[1] := ALINK800BURF[1] + t_T1_SS7.FieldByName('Total
2207  Inv').AsFloat;
2208                                ALINK800BURF[2] := ALINK800BURF[2] + t_T1_SS7.FieldByName('VS
2209  Inv').AsFloat;
2210                              end;
2211
2212                        //RFPC0018 A link for database 800 underground cable
2213                        if GetSS7Temp( RCLLI, 'Clink_UndFInv')
2214                         then begin
2215                                ALINK800UNDF[1] := ALINK800UNDF[1] + t_T1_SS7.FieldByName('Total
2216  Inv').AsFloat;
```

```
2217                                ALINK800UNDF[2] := ALINK800UNDF[2] + t_T1_SS7.FieldByName('VS
2218    Inv').AsFloat;
2219                              end;
2220
2221                            //RFPC0018 A link for database 800 poles
2222                            if GetSS7Temp( RCLLI, 'Clink_PoleInv')
2223                             then begin
2224                                ALINK800POLE[1] := ALINK800POLE[1] + t_T1_SS7.FieldByName('Total
2225    Inv').AsFloat;
2226                                ALINK800POLE[2] := ALINK800POLE[2] + t_T1_SS7.FieldByName('VS
2227    Inv').AsFloat;
2228                              end;
2229
2230                            //RFPC0018 A link for database 800 conduit
2231                            if GetSS7Temp( RCLLI, 'Clink_CondInv')
2232                             then begin
2233                                ALINK800COND[1] := ALINK800COND[1] + t_T1_SS7.FieldByName('Total
2234    Inv').AsFloat;
2235                                ALINK800COND[2] := ALINK800COND[2] + t_T1_SS7.FieldByName('VS
2236    Inv').AsFloat;
2237                              end;
2238
2239                            //RFPC0018 ------------------------------------------
2240                            //RFPC0018 A link for database 800 secondary SCP2
2241                            RCLLI := dm_ICM.t_SS7_Location.FieldByName('DB800_SecSCP2').AsString;
2242
2243                            //RFPC0018 A link for database 800 circuit equipment
2244                            if GetSS7Temp( RCLLI, 'Clink_CktInv')
2245                             then begin
2246                                ALINK800CKT[1] := ALINK800CKT[1] + t_T1_SS7.FieldByName('Total
2247    Inv').AsFloat;
2248                                ALINK800CKT[2] := ALINK800CKT[2] + t_T1_SS7.FieldByName('VS Inv').AsFloat;
2249                              end;
2250
2251                            //RFPC0018 A link for database 800 circuit equipment
2252                            if GetSS7Temp( RCLLI, 'STPSpec_CktInv')
2253                             then begin
2254                                ALINK800CKT[1] := ALINK800CKT[1] + t_T1_SS7.FieldByName('Total
2255    Inv').AsFloat;
```

```
2256                                ALINK800CKT[2] := ALINK800CKT[2] + t_T1_SS7.FieldByName('VS Inv').AsFloat;
2257                              end;
2258
2259                            //RFPC0018 A link for database 800 aerial cable
2260                            if GetSS7Temp( RCLLI, 'Clink_AerFInv')
2261                             then begin
2262                                ALINK800AERF[1] := ALINK800AERF[1] + t_T1_SS7.FieldByName('Total
2263  Inv').AsFloat;
2264                                ALINK800AERF[2] := ALINK800AERF[2] + t_T1_SS7.FieldByName('VS
2265  Inv').AsFloat;
2266                              end;
2267
2268                            //RFPC0018 A link for database 800 buried cable
2269                            if GetSS7Temp( RCLLI, 'Clink_BurFInv')
2270                             then begin
2271                                ALINK800BURF[1] := ALINK800BURF[1] + t_T1_SS7.FieldByName('Total
2272  Inv').AsFloat;
2273                                ALINK800BURF[2] := ALINK800BURF[2] + t_T1_SS7.FieldByName('VS
2274  Inv').AsFloat;
2275                              end;
2276
2277                            //RFPC0018 A link for database 800 underground cable
2278                            if GetSS7Temp( RCLLI, 'Clink_UndFInv')
2279                             then begin
2280                                ALINK800UNDF[1] := ALINK800UNDF[1] + t_T1_SS7.FieldByName('Total
2281  Inv').AsFloat;
2282                                ALINK800UNDF[2] := ALINK800UNDF[2] + t_T1_SS7.FieldByName('VS
2283  Inv').AsFloat;
2284                              end;
2285
2286                            //RFPC0018 A link for database 800 poles
2287                            if GetSS7Temp( RCLLI, 'Clink_PoleInv')
2288                             then begin
2289                                ALINK800POLE[1] := ALINK800POLE[1] + t_T1_SS7.FieldByName('Total
2290  Inv').AsFloat;
2291                                ALINK800POLE[2] := ALINK800POLE[2] + t_T1_SS7.FieldByName('VS
2292  Inv').AsFloat;
2293                              end;
2294
```

```
2295                        //RFPC0018 A link for database 800 conduit
2296                        if GetSS7Temp( RCLLI, 'Clink_CondInv')
2297                         then begin
2298                                ALINK800COND[1] := ALINK800COND[1] + t_T1_SS7.FieldByName('Total
2299   Inv').AsFloat;
2300                                ALINK800COND[2] := ALINK800COND[2] + t_T1_SS7.FieldByName('VS
2301   Inv').AsFloat;
2302                             end;
2303
2304                        //RFPC0018 ------------------------------------------
2305                        //RFPC0018 A link for LIDB primary SCP1
2306                        RCLLI := dm_ICM.t_SS7_Location.FieldByName('LIDB_PriSCP1').AsString;
2307
2308                        //RFPC0018 A link for LIDB circuit equipment
2309                        if GetSS7Temp( RCLLI, 'Clink_CktInv')
2310                         then begin
2311                                ALINKLIDBCKT[1] := ALINKLIDBCKT[1] + t_T1_SS7.FieldByName('Total
2312   Inv').AsFloat;
2313                                ALINKLIDBCKT[2] := ALINKLIDBCKT[2] + t_T1_SS7.FieldByName('VS
2314   Inv').AsFloat;
2315                             end;
2316
2317                        //RFPC0018 A link for LIDB circuit equipment
2318                        if GetSS7Temp( RCLLI, 'STPSpec_CktInv')
2319                         then begin
2320                                ALINKLIDBCKT[1] := ALINKLIDBCKT[1] + t_T1_SS7.FieldByName('Total
2321   Inv').AsFloat;
2322                                ALINKLIDBCKT[2] := ALINKLIDBCKT[2] + t_T1_SS7.FieldByName('VS
2323   Inv').AsFloat;
2324                             end;
2325
2326                        //RFPC0018 A link for LIDB aerial cable
2327                        if GetSS7Temp( RCLLI, 'Clink_AerFInv')
2328                         then begin
2329                                ALINKLIDBAERF[1] := ALINKLIDBAERF[1] + t_T1_SS7.FieldByName('Total
2330   Inv').AsFloat;
2331                                ALINKLIDBAERF[2] := ALINKLIDBAERF[2] + t_T1_SS7.FieldByName('VS
2332   Inv').AsFloat;
2333                             end;
```

```
2334
2335                        //RFPC0018 A link for LIDB buried cable
2336                        if GetSS7Temp( RCLLI, 'Clink_BurFInv')
2337                         then begin
2338                                ALINKLIDBBURF[1] := ALINKLIDBBURF[1] + t_T1_SS7.FieldByName('Total
2339 Inv').AsFloat;
2340                                ALINKLIDBBURF[2] := ALINKLIDBBURF[2] + t_T1_SS7.FieldByName('VS
2341 Inv').AsFloat;
2342                             end;
2343
2344                        //RFPC0018 A link for LIDB underground cable
2345                        if GetSS7Temp( RCLLI, 'Clink_UndFInv')
2346                         then begin
2347                                ALINKLIDBUNDF[1] := ALINKLIDBUNDF[1] + t_T1_SS7.FieldByName('Total
2348 Inv').AsFloat;
2349                                ALINKLIDBUNDF[2] := ALINKLIDBUNDF[2] + t_T1_SS7.FieldByName('VS
2350 Inv').AsFloat;
2351                             end;
2352
2353                        //RFPC0018 A link for LIDB poles
2354                        if GetSS7Temp( RCLLI, 'Clink_PoleInv')
2355                         then begin
2356                                ALINKLIDBPOLE[1] := ALINKLIDBPOLE[1] + t_T1_SS7.FieldByName('Total
2357 Inv').AsFloat;
2358                                ALINKLIDBPOLE[2] := ALINKLIDBPOLE[2] + t_T1_SS7.FieldByName('VS
2359 Inv').AsFloat;
2360                             end;
2361
2362                        //RFPC0018 A link for LIDB conduit
2363                        if GetSS7Temp( RCLLI, 'Clink_CondInv')
2364                         then begin
2365                                ALINKLIDBCOND[1] := ALINKLIDBCOND[1] + t_T1_SS7.FieldByName('Total
2366 Inv').AsFloat;
2367                                ALINKLIDBCOND[2] := ALINKLIDBCOND[2] + t_T1_SS7.FieldByName('VS
2368 Inv').AsFloat;
2369                             end;
2370
2371                        //RFPC0018 ----------------------------------------
2372                        //RFPC0018 A link for LIDB primary SCP2
```

```
2373                        RCLLI := dm_ICM.t_SS7_Location.FieldByName('LIDB_PriSCP2').AsString;
2374
2375                        //RFPC0018 A link for LIDB circuit equipment
2376                        if GetSS7Temp( RCLLI, 'Clink_CktInv')
2377                         then begin
2378                                ALINKLIDBCKT[1] := ALINKLIDBCKT[1] + t_T1_SS7.FieldByName('Total
2379  Inv').AsFloat;
2380                                ALINKLIDBCKT[2] := ALINKLIDBCKT[2] + t_T1_SS7.FieldByName('VS
2381  Inv').AsFloat;
2382                             end;
2383
2384                        //RFPC0018 A link for LIDB circuit equipment
2385                        if GetSS7Temp( RCLLI, 'STPSpec_CktInv')
2386                         then begin
2387                                ALINKLIDBCKT[1] := ALINKLIDBCKT[1] + t_T1_SS7.FieldByName('Total
2388  Inv').AsFloat;
2389                                ALINKLIDBCKT[2] := ALINKLIDBCKT[2] + t_T1_SS7.FieldByName('VS
2390  Inv').AsFloat;
2391                             end;
2392
2393                        //RFPC0018 A link for LIDB aerial cable
2394                        if GetSS7Temp( RCLLI, 'Clink_AerFInv')
2395                         then begin
2396                                ALINKLIDBAERF[1] := ALINKLIDBAERF[1] + t_T1_SS7.FieldByName('Total
2397  Inv').AsFloat;
2398                                ALINKLIDBAERF[2] := ALINKLIDBAERF[2] + t_T1_SS7.FieldByName('VS
2399  Inv').AsFloat;
2400                             end;
2401
2402                        //RFPC0018 A link for LIDB buried cable
2403                        if GetSS7Temp( RCLLI, 'Clink_BurFInv')
2404                         then begin
2405                                ALINKLIDBBURF[1] := ALINKLIDBBURF[1] + t_T1_SS7.FieldByName('Total
2406  Inv').AsFloat;
2407                                ALINKLIDBBURF[2] := ALINKLIDBBURF[2] + t_T1_SS7.FieldByName('VS
2408  Inv').AsFloat;
2409                             end;
2410
2411                        //RFPC0018 A link for LIDB underground cable
```

```
2412                        if GetSS7Temp( RCLLI, 'Clink_UndFInv')
2413                         then begin
2414                                ALINKLIDBUNDF[1] := ALINKLIDBUNDF[1] + t_T1_SS7.FieldByName('Total
2415  Inv').AsFloat;
2416                                ALINKLIDBUNDF[2] := ALINKLIDBUNDF[2] + t_T1_SS7.FieldByName('VS
2417  Inv').AsFloat;
2418                             end;
2419
2420                        //RFPC0018 A link for LIDB poles
2421                        if GetSS7Temp( RCLLI, 'Clink_PoleInv')
2422                         then begin
2423                                ALINKLIDBPOLE[1] := ALINKLIDBPOLE[1] + t_T1_SS7.FieldByName('Total
2424  Inv').AsFloat;
2425                                ALINKLIDBPOLE[2] := ALINKLIDBPOLE[2] + t_T1_SS7.FieldByName('VS
2426  Inv').AsFloat;
2427                             end;
2428
2429                        //RFPC0018 A link for LIDB conduit
2430                        if GetSS7Temp( RCLLI, 'Clink_CondInv')
2431                         then begin
2432                                ALINKLIDBCOND[1] := ALINKLIDBCOND[1] + t_T1_SS7.FieldByName('Total
2433  Inv').AsFloat;
2434                                ALINKLIDBCOND[2] := ALINKLIDBCOND[2] + t_T1_SS7.FieldByName('VS
2435  Inv').AsFloat;
2436                             end;
2437
2438                        //RFPC0018 ----------------------------------------
2439                        //RFPC0018 A link for LIDB secondary SCP1
2440                        RCLLI := dm_ICM.t_SS7_Location.FieldByName('LIDB_SecSCP1').AsString;
2441
2442                        //RFPC0018 A link for LIDB circuit equipment
2443                        if GetSS7Temp( RCLLI, 'Clink_CktInv')
2444                         then begin
2445                                ALINKLIDBCKT[1] := ALINKLIDBCKT[1] + t_T1_SS7.FieldByName('Total
2446  Inv').AsFloat;
2447                                ALINKLIDBCKT[2] := ALINKLIDBCKT[2] + t_T1_SS7.FieldByName('VS
2448  Inv').AsFloat;
2449                             end;
2450
```

```
2451                        //RFPC0018 A link for LIDB circuit equipment
2452                        if GetSS7Temp( RCLLI, 'STPSpec_CktInv')
2453                         then begin
2454                                ALINKLIDBCKT[1] := ALINKLIDBCKT[1] + t_T1_SS7.FieldByName('Total
2455   Inv').AsFloat;
2456                                ALINKLIDBCKT[2] := ALINKLIDBCKT[2] + t_T1_SS7.FieldByName('VS
2457   Inv').AsFloat;
2458                             end;
2459
2460                        //RFPC0018 A link for LIDB aerial cable
2461                        if GetSS7Temp( RCLLI, 'Clink_AerFInv')
2462                         then begin
2463                                ALINKLIDBAERF[1] := ALINKLIDBAERF[1] + t_T1_SS7.FieldByName('Total
2464   Inv').AsFloat;
2465                                ALINKLIDBAERF[2] := ALINKLIDBAERF[2] + t_T1_SS7.FieldByName('VS
2466   Inv').AsFloat;
2467                             end;
2468
2469                        //RFPC0018 A link for LIDB buried cable
2470                        if GetSS7Temp( RCLLI, 'Clink_BurFInv')
2471                         then begin
2472                                ALINKLIDBBURF[1] := ALINKLIDBBURF[1] + t_T1_SS7.FieldByName('Total
2473   Inv').AsFloat;
2474                                ALINKLIDBBURF[2] := ALINKLIDBBURF[2] + t_T1_SS7.FieldByName('VS
2475   Inv').AsFloat;
2476                             end;
2477
2478                        //RFPC0018 A link for LIDB underground cable
2479                        if GetSS7Temp( RCLLI, 'Clink_UndFInv')
2480                         then begin
2481                                ALINKLIDBUNDF[1] := ALINKLIDBUNDF[1] + t_T1_SS7.FieldByName('Total
2482   Inv').AsFloat;
2483                                ALINKLIDBUNDF[2] := ALINKLIDBUNDF[2] + t_T1_SS7.FieldByName('VS
2484   Inv').AsFloat;
2485                             end;
2486
2487                        //RFPC0018 A link for LIDB poles
2488                        if GetSS7Temp( RCLLI, 'Clink_PoleInv')
2489                         then begin
```

```
2490                                ALINKLIDBPOLE[1] := ALINKLIDBPOLE[1] + t_T1_SS7.FieldByName('Total
2491    Inv').AsFloat;
2492                                ALINKLIDBPOLE[2] := ALINKLIDBPOLE[2] + t_T1_SS7.FieldByName('VS
2493    Inv').AsFloat;
2494                               end;
2495
2496                            //RFPC0018 A link for LIDB conduit
2497                            if GetSS7Temp( RCLLI, 'Clink_CondInv')
2498                             then begin
2499                                ALINKLIDBCOND[1] := ALINKLIDBCOND[1] + t_T1_SS7.FieldByName('Total
2500    Inv').AsFloat;
2501                                ALINKLIDBCOND[2] := ALINKLIDBCOND[2] + t_T1_SS7.FieldByName('VS
2502    Inv').AsFloat;
2503                               end;
2504
2505                            //RFPC0018 ----------------------------------------
2506                            //RFPC0018 A link for LIDB secondary SCP2
2507                            RCLLI := dm_ICM.t_SS7_Location.FieldByName('LIDB_SecSCP2').AsString;
2508
2509                            //RFPC0018 A link for LIDB circuit equipment
2510                            if GetSS7Temp( RCLLI, 'Clink_CktInv')
2511                             then begin
2512                                ALINKLIDBCKT[1] := ALINKLIDBCKT[1] + t_T1_SS7.FieldByName('Total
2513    Inv').AsFloat;
2514                                ALINKLIDBCKT[2] := ALINKLIDBCKT[2] + t_T1_SS7.FieldByName('VS
2515    Inv').AsFloat;
2516                               end;
2517
2518                            //RFPC0018 A link for LIDB circuit equipment
2519                            if GetSS7Temp( RCLLI, 'STPSpec_CktInv')
2520                             then begin
2521                                ALINKLIDBCKT[1] := ALINKLIDBCKT[1] + t_T1_SS7.FieldByName('Total
2522    Inv').AsFloat;
2523                                ALINKLIDBCKT[2] := ALINKLIDBCKT[2] + t_T1_SS7.FieldByName('VS
2524    Inv').AsFloat;
2525                               end;
2526
2527                            //RFPC0018 A link for LIDB aerial cable
2528                            if GetSS7Temp( RCLLI, 'Clink_AerFInv')
```

```
2529                          then begin
2530                                ALINKLIDBAERF[1] := ALINKLIDBAERF[1] + t_T1_SS7.FieldByName('Total
2531   Inv').AsFloat;
2532                                ALINKLIDBAERF[2] := ALINKLIDBAERF[2] + t_T1_SS7.FieldByName('VS
2533   Inv').AsFloat;
2534                           end;
2535
2536                         //RFPC0018 A link for LIDB buried cable
2537                         if GetSS7Temp( RCLLI, 'Clink_BurFInv')
2538                          then begin
2539                                ALINKLIDBBURF[1] := ALINKLIDBBURF[1] + t_T1_SS7.FieldByName('Total
2540   Inv').AsFloat;
2541                                ALINKLIDBBURF[2] := ALINKLIDBBURF[2] + t_T1_SS7.FieldByName('VS
2542   Inv').AsFloat;
2543                           end;
2544
2545                         //RFPC0018 A link for LIDB underground cable
2546                         if GetSS7Temp( RCLLI, 'Clink_UndFInv')
2547                          then begin
2548                                ALINKLIDBUNDF[1] := ALINKLIDBUNDF[1] + t_T1_SS7.FieldByName('Total
2549   Inv').AsFloat;
2550                                ALINKLIDBUNDF[2] := ALINKLIDBUNDF[2] + t_T1_SS7.FieldByName('VS
2551   Inv').AsFloat;
2552                           end;
2553
2554                         //RFPC0018 A link for LIDB poles
2555                         if GetSS7Temp( RCLLI, 'Clink_PoleInv')
2556                          then begin
2557                                ALINKLIDBPOLE[1] := ALINKLIDBPOLE[1] + t_T1_SS7.FieldByName('Total
2558   Inv').AsFloat;
2559                                ALINKLIDBPOLE[2] := ALINKLIDBPOLE[2] + t_T1_SS7.FieldByName('VS
2560   Inv').AsFloat;
2561                           end;
2562
2563                         //RFPC0018 A link for LIDB conduit
2564                         if GetSS7Temp( RCLLI, 'Clink_CondInv')
2565                          then begin
2566                                ALINKLIDBCOND[1] := ALINKLIDBCOND[1] + t_T1_SS7.FieldByName('Total
2567   Inv').AsFloat;
```

```
2568                                ALINKLIDBCOND[2] := ALINKLIDBCOND[2] + t_T1_SS7.FieldByName('VS
2569  Inv').AsFloat;
2570                              end;
2571
2572                          //RFPC0018 average the investments
2573                          for I:= 1 to 2 do
2574                            begin
2575                              ALINK800CKT[I]    := ALINK800CKT[I] / 8.0;
2576                              ALINK800AERF[I]   := ALINK800AERF[I] / 8.0;
2577                              ALINK800BURF[I]   := ALINK800BURF[I] / 8.0;
2578                              ALINK800UNDF[I]   := ALINK800UNDF[I] / 8.0;
2579                              ALINK800POLE[I]   := ALINK800POLE[I] / 8.0;
2580                              ALINK800COND[I]   := ALINK800COND[I] / 8.0;
2581
2582                              ALINKLIDBCKT[I]   := ALINKLIDBCKT[I] / 8.0;
2583                              ALINKLIDBAERF[I]  := ALINKLIDBAERF[I] / 8.0;
2584                              ALINKLIDBBURF[I]  := ALINKLIDBBURF[I] / 8.0;
2585                              ALINKLIDBUNDF[I]  := ALINKLIDBUNDF[I] / 8.0;
2586                              ALINKLIDBPOLE[I]  := ALINKLIDBPOLE[I] / 8.0;
2587                              ALINKLIDBCOND[I]  := ALINKLIDBCOND[I] / 8.0;
2588                            end;
2589                        end;
2590                end;
2591          //RFPC0018 Section VII part 7 end
2592
2593          //RFPC0018 Section VII Part 8 begin
2594      {
2595           8.  GTT Software Investment per TCAP Message at Regional and Database STPs
2596
2597               Global Title Translation (GTT) software is required at regional STPs
2598               for TCAP messages.  Total investment value is from the STP Investment
2599               Table, and the capacity per port is based upon the number of ports
2600               from the STP Table and the estimated message capacity per port from
2601               the Network Parameters Table.
2602
2603            Total GTT Investment per Regional STP:
2604
2605               GTTInv_R  = SS7_Invst.Investment * (1 + STPLoading_SW)
2606               where SS7_Invst.Item = "STP_GTTSW" and STP_Table.Switch_Type =
```

```
2607            STP_Table.STP_Type for STP_CLLI = RSTP1
2608
2609          Estimated Message Capacity and Demand per Port:
2610
2611           GTTMsgCap = Value from Network Parameters Table
2612            GTTMsgDem = DLinkUtil * GTTMsgCap,
2613            where DLinkUtil is value from Network Parameters Table.
2614
2615          GTT Investment per TCAP Message at Regional and Database Regional STPs:
2616
2617          Average Investments:
2618            GTTInv_R / (GTTMsgDem * STP_Table.Ports_InSvc)
2619              if local STP_CLLI = RSTP1 or RSTP2
2620
2621            TCAPSWR =(2*GTTInv_R) / (GTTMsgDem * STP_Table.Ports_InSvc)
2622               otherwise (Use Ports_InSvc of STP_CLLI = RSTP1)
2623
2624          Volume-Sensitive Investments:
2625             GTTInv_R / (GTTMsgCap*STP_Table.Ports_InSvc)
2626             if local STP_CLLI = RSTP1 or RSTP2
2627
2628               TCAPSWR =(2*GTTInv_R) / (GTTMsgCap * STP_Table.Ports_InSvc)
2629                       otherwise (Use Ports_InSvc of STP_CLLI = RSTP1)
2630    }
2631
2632        RCLLI := dm_ICM.t_SS7_STP.FieldByName('Regional_STP1').AsString;
2633        if CLLI = RCLLI
2634          then begin
2635                 if GetSS7Temp(CLLI, 'STP_GTTSW')
2636                   then begin
2637                          GTTInv_R[1] := t_T1_SS7.FieldByName('Total Inv').AsFloat;
2638                          GTTInv_R[2] := t_T1_SS7.FieldByName('VS Inv').AsFloat;
2639                        end;
2640               end
2641          else begin
2642                 GTTInv_R[1] := 0.0;
2643                 GTTInv_R[2] := 0.0;
2644              end;
2645        RCLLI := dm_ICM.t_SS7_STP.FieldByName('Regional_STP2').AsString;
```

```
2646        if CLLI = RCLLI
2647          then begin
2648                 if GetSS7Temp(CLLI, 'STP_GTTSW')
2649                   then begin
2650                          GTTInv_R[1] := GTTInv_R[1] + t_T1_SS7.FieldByName('Total Inv').AsFloat;
2651                          GTTInv_R[2] := GTTInv_R[2] + t_T1_SS7.FieldByName('VS Inv').AsFloat;
2652                        end;
2653               end;
2654
2655        GTTMsgCap := GetParameter('GTTMsgCap');
2656        GTTMsgDem := GTTMsgCap * GetParameter('DLinkUtil');
2657
2658        if (CLLI=RCLLI) or (CLLI=dm_ICM.t_SS7_STP.FieldByName('Regional_STP2').AsString)
2659         then begin
2660                if GTTMsgDem > 0.0
2661                   then TCAPSWR[1] := GTTInv_R[1] / (GTTMsgDem *
2662  dm_ICM.t_SS7_STP.FieldByName('Ports_InSvc').AsFloat)
2663                   else TCAPSWR[1] := 0.0;
2664                if GTTMsgCap > 0.0
2665                   then TCAPSWR[2] := GTTInv_R[2] / (GTTMsgCap *
2666  dm_ICM.t_SS7_STP.FieldByName('Ports_InSvc').AsFloat)
2667                   else TCAPSWR[2] := 0.0;
2668              end
2669         else begin //RFPC0018 load ports inservice from Regional_STP1
2670                CurrentRec := dm_ICM.t_SS7_STP.GetBookMark;
2671                if dm_ICM.t_SS7_STP.FindKey([RCLLI])
2672                  then X := dm_ICM.t_SS7_STP.FieldByName('Ports_InSvc').AsFloat
2673                  else begin
2674                         X := 0.0;
2675                         MessageDlg('Could not find ports inservice for '+ RCLLI + '.', mtError, [mbok], 0);
2676                         WeAreWatching.LogEvent(1,'Process SS7','Could not find ports inservice for '+ RCLLI
2677  + '.');
2678                       end;
2679                dm_ICM.t_SS7_STP.GotoBookMark(CurrentRec);
2680                dm_ICM.t_SS7_STP.FreeBookMark(CurrentRec);
2681
2682                if (GTTMsgDem*X) > 0.0
2683                  then TCAPSWR[1] := (2.0 * GTTInv_R[1]) / (GTTMsgDem * X)
2684                  else TCAPSWR[1] := 0.0;
```

```
2685                if (GTTMsgCap*X) > 0.0
2686                  then TCAPSWR[2] := (2.0 * GTTInv_R[2]) / (GTTMsgCap * X)
2687                  else TCAPSWR[2] := 0.0;
2688              end;
2689    //RFPC0018 Section VII Part 8 end
2690
2691    //RFPC0018 Section VII Part 9 begin
2692    {
2693          9.  Investment per Regional STP Port
2694
2695          Each STP requires an investment for the port at the regional STP,
2696          which is the average of the port investments at regional STP pair
2697          RTSP1 and RSTP2.  For the variables X below, Xi refers to the port
2698          investment value stored for RSTPi (i = 1, 2).  Set each result
2699          below to 0 if STP_CLLI being studied is equal to either RSTP1 or RSTP2.
2700
2701          RSPORTSWT  = (STPPORTSWT1 + STPPORTSWT2) / 2
2702          RSPORTCKT  = (STPPORTCKT1 + STPPORTCKT2) / 2
2703          RSPORTAERF = (STPPORTAERF1 + STPPORTAERF2) / 2
2704          RSPORTBURF = (STPPORTBURF1 + STPPORTBURF2) / 2
2705          RSPORTUNDF = (STPPORTUNDF1 + STPPORTUNDF2) / 2
2706          RSPORTPOLE = (STPPORTPOLE1 + STPPORTPOLE2) / 2
2707          RSPORTCOND = (STPPORTCOND1 + STPPORTCOND2) / 2
2708    }
2709        if (CLLI <> dm_ICM.t_SS7_STP.FieldByName('Regional_STP1').AsString) and
2710           (CLLI <> dm_ICM.t_SS7_STP.FieldByName('Regional_STP2').AsString)
2711          then begin
2712                 //RFPC0018 ------------------------------------------
2713                 //RFPC0018 process data for regional stp port - RSTP1
2714                 RCLLI := dm_ICM.t_SS7_STP.FieldByName('Regional_STP1').AsString;
2715
2716                 //RFPC0018 regional stp port circuit equipment
2717                 if GetSS7Temp( RCLLI, 'STPPORTSWT')
2718                  then begin
2719                         RSPORTSWT[1] := RSPORTSWT[1] + t_T1_SS7.FieldByName('Total Inv').AsFloat;
2720                         RSPORTSWT[2] := RSPORTSWT[2] + t_T1_SS7.FieldByName('VS Inv').AsFloat;
2721                       end;
2722
2723                 //RFPC0018 regional stp port circuit equipment
```

```
2724                if GetSS7Temp( RCLLI, 'STPPORTCKT')
2725                 then begin
2726                        RSPORTCKT[1] := RSPORTCKT[1] + t_T1_SS7.FieldByName('Total Inv').AsFloat;
2727                        RSPORTCKT[2] := RSPORTCKT[2] + t_T1_SS7.FieldByName('VS Inv').AsFloat;
2728                      end;
2729
2730                //RFPC0018 regional stp port aerial cable
2731                if GetSS7Temp( RCLLI, 'STPPORTAERF')
2732                 then begin
2733                        RSPORTAERF[1] := RSPORTAERF[1] + t_T1_SS7.FieldByName('Total Inv').AsFloat;
2734                        RSPORTAERF[2] := RSPORTAERF[2] + t_T1_SS7.FieldByName('VS Inv').AsFloat;
2735                      end;
2736
2737                //RFPC0018 Aregional stp port buried cable
2738                if GetSS7Temp( RCLLI, 'STPPORTBURF')
2739                 then begin
2740                        RSPORTBURF[1] := RSPORTBURF[1] + t_T1_SS7.FieldByName('Total Inv').AsFloat;
2741                        RSPORTBURF[2] := RSPORTBURF[2] + t_T1_SS7.FieldByName('VS Inv').AsFloat;
2742                      end;
2743
2744                //RFPC0018 A regional stp port underground cable
2745                if GetSS7Temp( RCLLI, 'STPPORTUNDF')
2746                 then begin
2747                        RSPORTUNDF[1] := RSPORTUNDF[1] + t_T1_SS7.FieldByName('Total Inv').AsFloat;
2748                        RSPORTUNDF[2] := RSPORTUNDF[2] + t_T1_SS7.FieldByName('VS Inv').AsFloat;
2749                      end;
2750
2751                //RFPC0018 regional stp port poles
2752                if GetSS7Temp( RCLLI, 'STPPORTPOLE')
2753                 then begin
2754                        RSPORTPOLE[1] := RSPORTPOLE[1] + t_T1_SS7.FieldByName('Total Inv').AsFloat;
2755                        RSPORTPOLE[2] := RSPORTPOLE[2] + t_T1_SS7.FieldByName('VS Inv').AsFloat;
2756                      end;
2757
2758                //RFPC0018 regional stp port conduit
2759                if GetSS7Temp( RCLLI, 'STPPORTCOND')
2760                 then begin
2761                        RSPORTCOND[1] := RSPORTCOND[1] + t_T1_SS7.FieldByName('Total Inv').AsFloat;
2762                        RSPORTCOND[2] := RSPORTCOND[2] + t_T1_SS7.FieldByName('VS Inv').AsFloat;
```

```
2763                    end;
2764
2765                //RFPC0018 ------------------------------------------
2766                //RFPC0018 process data for regional stp port - RSTP2
2767                RCLLI := dm_ICM.t_SS7_STP.FieldByName('Regional_STP2').AsString;
2768
2769                //RFPC0018 regional stp port circuit equipment
2770                if GetSS7Temp( RCLLI, 'STPPORTSWT')
2771                 then begin
2772                        RSPORTSWT[1] := RSPORTSWT[1] + t_T1_SS7.FieldByName('Total Inv').AsFloat;
2773                        RSPORTSWT[2] := RSPORTSWT[2] + t_T1_SS7.FieldByName('VS Inv').AsFloat;
2774                      end;
2775
2776                //RFPC0018 regional stp port circuit equipment
2777                if GetSS7Temp( RCLLI, 'STPPORTCKT')
2778                 then begin
2779                        RSPORTCKT[1] := RSPORTCKT[1] + t_T1_SS7.FieldByName('Total Inv').AsFloat;
2780                        RSPORTCKT[2] := RSPORTCKT[2] + t_T1_SS7.FieldByName('VS Inv').AsFloat;
2781                      end;
2782
2783                //RFPC0018 regional stp port aerial cable
2784                if GetSS7Temp( RCLLI, 'STPPORTAERF')
2785                 then begin
2786                        RSPORTAERF[1] := RSPORTAERF[1] + t_T1_SS7.FieldByName('Total Inv').AsFloat;
2787                        RSPORTAERF[2] := RSPORTAERF[2] + t_T1_SS7.FieldByName('VS Inv').AsFloat;
2788                      end;
2789
2790                //RFPC0018 Aregional stp port buried cable
2791                if GetSS7Temp( RCLLI, 'STPPORTBURF')
2792                 then begin
2793                        RSPORTBURF[1] := RSPORTBURF[1] + t_T1_SS7.FieldByName('Total Inv').AsFloat;
2794                        RSPORTBURF[2] := RSPORTBURF[2] + t_T1_SS7.FieldByName('VS Inv').AsFloat;
2795                      end;
2796
2797                //RFPC0018 A regional stp port underground cable
2798                if GetSS7Temp( RCLLI, 'STPPORTUNDF')
2799                 then begin
2800                        RSPORTUNDF[1] := RSPORTUNDF[1] + t_T1_SS7.FieldByName('Total Inv').AsFloat;
2801                        RSPORTUNDF[2] := RSPORTUNDF[2] + t_T1_SS7.FieldByName('VS Inv').AsFloat;
```

```
2802                    end;
2803
2804                //RFPC0018 regional stp port poles
2805                if GetSS7Temp( RCLLI, 'STPPORTPOLE')
2806                 then begin
2807                       RSPORTPOLE[1] := RSPORTPOLE[1] + t_T1_SS7.FieldByName('Total Inv').AsFloat;
2808                       RSPORTPOLE[2] := RSPORTPOLE[2] + t_T1_SS7.FieldByName('VS Inv').AsFloat;
2809                    end;
2810
2811                //RFPC0018 regional stp port conduit
2812                if GetSS7Temp( RCLLI, 'STPPORTCOND')
2813                 then begin
2814                       RSPORTCOND[1] := RSPORTCOND[1] + t_T1_SS7.FieldByName('Total Inv').AsFloat;
2815                       RSPORTCOND[2] := RSPORTCOND[2] + t_T1_SS7.FieldByName('VS Inv').AsFloat;
2816                    end;
2817
2818                for I := 1 to 2 do
2819                  begin
2820                    RSPORTSWT[I] := RSPORTSWT[I] / 2.0;
2821                    RSPORTCKT[I] := RSPORTCKT[I] / 2.0;
2822                    RSPORTAERF[I] := RSPORTAERF[I] / 2.0;
2823                    RSPORTBURF[I] := RSPORTBURF[I] / 2.0;
2824                    RSPORTUNDF[I] := RSPORTUNDF[I] / 2.0;
2825                    RSPORTPOLE[I] := RSPORTPOLE[I] / 2.0;
2826                    RSPORTCOND[I] := RSPORTCOND[I] / 2.0;
2827                  end;
2828              end;
2829  //RFPC0018 Section VII Part 9 end
2830
2831  //RFPC0018 Section VII Part 10 begin
2832   {
2833        Each STP requires an investment for the ports at the regional STPs
2834        colocated with DB800 SCPs, which is the average of the port investments
2835        at the primary and secondary pairs of regional STPs for 800 database
2836        service.  For the variables X below, Xpi refers to the port investment
2837        value stored for primary 800 STPi (i = 1, 2), and Xsi refers to the port
2838        investment value stored for secondary STPi (i = 1, 2).  The primary 800
2839        STPs are identified by the CLLI Codes Reg_Location.DB800_PriSCP1 and
2840        Reg_Location.DB800_PriSCP2, and secondary 800 STPs are identified by the
```

```
2841          CLLI Codes Reg_Location.DB800_SecSCP1 and Reg_Location.DB800_SecSCP2,
2842          where Reg_Location.RegSTP1 = RSTP1.
2843      }
2844         //RFPC0018 database 800 ports
2845         RCLLI := dm_ICM.t_SS7_STP.FieldByName('Regional_STP1').AsString;
2846         if dm_ICM.t_SS7_Location.FindKey([RCLLI])
2847           then begin
2848                  //RFPC0018 software
2849                  if (RCLLI <> dm_ICM.t_SS7_Location.FieldByName('DB800_PriSCP1').AsString) and
2850                     (RCLLI <> dm_ICM.t_SS7_Location.FieldByName('DB800_PriSCP2').AsString)
2851                    then if GetSS7Temp(dm_ICM.t_SS7_Location.FieldByName('DB800_PriSCP1').AsString,
2852   'STPPORTSWT')
2853                           then begin
2854                                  DB800PORTSWT[1] := DB800PORTSWT[1] + t_T1_SS7.FieldByName('Total
2855   Inv').AsFloat;
2856                                  DB800PORTSWT[2] := DB800PORTSWT[2] + t_T1_SS7.FieldByName('VS Inv').AsFloat;
2857                                end;
2858                  if (RCLLI <> dm_ICM.t_SS7_Location.FieldByName('DB800_SecSCP1').AsString) and
2859                     (RCLLI <> dm_ICM.t_SS7_Location.FieldByName('DB800_SecSCP2').AsString)
2860                    then if GetSS7Temp(dm_ICM.t_SS7_Location.FieldByName('DB800_SecSCP1').AsString,
2861   'STPPORTSWT')
2862                           then begin
2863                                  DB800PORTSWT[1] := DB800PORTSWT[1] + t_T1_SS7.FieldByName('Total
2864   Inv').AsFloat;
2865                                  DB800PORTSWT[2] := DB800PORTSWT[2] + t_T1_SS7.FieldByName('VS Inv').AsFloat;
2866                                end;
2867
2868                  //RFPC0018 circuit
2869                  if (RCLLI <> dm_ICM.t_SS7_Location.FieldByName('DB800_PriSCP1').AsString) and
2870                     (RCLLI <> dm_ICM.t_SS7_Location.FieldByName('DB800_PriSCP2').AsString)
2871                    then if GetSS7Temp(dm_ICM.t_SS7_Location.FieldByName('DB800_PriSCP1').AsString,
2872   'STPPORTCKT')
2873                           then begin
2874                                  DB800PORTCKT[1] := DB800PORTCKT[1] + t_T1_SS7.FieldByName('Total
2875   Inv').AsFloat;
2876                                  DB800PORTCKT[2] := DB800PORTCKT[2] + t_T1_SS7.FieldByName('VS Inv').AsFloat;
2877                                end;
2878                  if (RCLLI <> dm_ICM.t_SS7_Location.FieldByName('DB800_SecSCP1').AsString) and
2879                     (RCLLI <> dm_ICM.t_SS7_Location.FieldByName('DB800_SecSCP2').AsString)
```

```
2880                    then if GetSS7Temp(dm_ICM.t_SS7_Location.FieldByName('DB800_SecSCP1').AsString,
2881    'STPPORTCKT')
2882                            then begin
2883                                    DB800PORTCKT[1] := DB800PORTCKT[1] + t_T1_SS7.FieldByName('Total
2884    Inv').AsFloat;
2885                                    DB800PORTCKT[2] := DB800PORTCKT[2] + t_T1_SS7.FieldByName('VS Inv').AsFloat;
2886                                end;
2887
2888                  //RFPC0018 arial cable
2889                  if (RCLLI <> dm_ICM.t_SS7_Location.FieldByName('DB800_PriSCP1').AsString) and
2890                     (RCLLI <> dm_ICM.t_SS7_Location.FieldByName('DB800_PriSCP2').AsString)
2891                    then if GetSS7Temp(dm_ICM.t_SS7_Location.FieldByName('DB800_PriSCP1').AsString,
2892    'STPPORTAERF')
2893                            then begin
2894                                    DB800PORTAERF[1] := DB800PORTAERF[1] + t_T1_SS7.FieldByName('Total
2895    Inv').AsFloat;
2896                                    DB800PORTAERF[2] := DB800PORTAERF[2] + t_T1_SS7.FieldByName('VS
2897    Inv').AsFloat;
2898                                end;
2899                  if (RCLLI <> dm_ICM.t_SS7_Location.FieldByName('DB800_SecSCP1').AsString) and
2900                     (RCLLI <> dm_ICM.t_SS7_Location.FieldByName('DB800_SecSCP2').AsString)
2901                    then if GetSS7Temp(dm_ICM.t_SS7_Location.FieldByName('DB800_SecSCP1').AsString,
2902    'STPPORTAERF')
2903                            then begin
2904                                    DB800PORTAERF[1] := DB800PORTAERF[1] + t_T1_SS7.FieldByName('Total
2905    Inv').AsFloat;
2906                                    DB800PORTAERF[2] := DB800PORTAERF[2] + t_T1_SS7.FieldByName('VS
2907    Inv').AsFloat;
2908                                end;
2909
2910                  //RFPC0018 buried cable
2911                  if (RCLLI <> dm_ICM.t_SS7_Location.FieldByName('DB800_PriSCP1').AsString) and
2912                     (RCLLI <> dm_ICM.t_SS7_Location.FieldByName('DB800_PriSCP2').AsString)
2913                    then if GetSS7Temp(dm_ICM.t_SS7_Location.FieldByName('DB800_PriSCP1').AsString,
2914    'STPPORTBURF')
2915                            then begin
2916                                    DB800PORTBURF[1] := DB800PORTBURF[1] + t_T1_SS7.FieldByName('Total
2917    Inv').AsFloat;
```

```
2918                                    DB800PORTBURF[2] := DB800PORTBURF[2] + t_T1_SS7.FieldByName('VS
2919    Inv').AsFloat;
2920                                end;
2921                  if (RCLLI <> dm_ICM.t_SS7_Location.FieldByName('DB800_SecSCP1').AsString) and
2922                     (RCLLI <> dm_ICM.t_SS7_Location.FieldByName('DB800_SecSCP2').AsString)
2923                    then if GetSS7Temp(dm_ICM.t_SS7_Location.FieldByName('DB800_SecSCP1').AsString,
2924    'STPPORTBURF')
2925                           then begin
2926                                    DB800PORTBURF[1] := DB800PORTBURF[1] + t_T1_SS7.FieldByName('Total
2927    Inv').AsFloat;
2928                                    DB800PORTBURF[2] := DB800PORTBURF[2] + t_T1_SS7.FieldByName('VS
2929    Inv').AsFloat;
2930                                end;
2931
2932                  //RFPC0018 underground cable
2933                  if (RCLLI <> dm_ICM.t_SS7_Location.FieldByName('DB800_PriSCP1').AsString) and
2934                     (RCLLI <> dm_ICM.t_SS7_Location.FieldByName('DB800_PriSCP2').AsString)
2935                    then if GetSS7Temp(dm_ICM.t_SS7_Location.FieldByName('DB800_PriSCP1').AsString,
2936    'STPPORTUNDF')
2937                           then begin
2938                                    DB800PORTUNDF[1] := DB800PORTUNDF[1] + t_T1_SS7.FieldByName('Total
2939    Inv').AsFloat;
2940                                    DB800PORTUNDF[2] := DB800PORTUNDF[2] + t_T1_SS7.FieldByName('VS
2941    Inv').AsFloat;
2942                                end;
2943                  if (RCLLI <> dm_ICM.t_SS7_Location.FieldByName('DB800_SecSCP1').AsString) and
2944                     (RCLLI <> dm_ICM.t_SS7_Location.FieldByName('DB800_SecSCP2').AsString)
2945                    then if GetSS7Temp(dm_ICM.t_SS7_Location.FieldByName('DB800_SecSCP1').AsString,
2946    'STPPORTUNDF')
2947                           then begin
2948                                    DB800PORTUNDF[1] := DB800PORTUNDF[1] + t_T1_SS7.FieldByName('Total
2949    Inv').AsFloat;
2950                                    DB800PORTUNDF[2] := DB800PORTUNDF[2] + t_T1_SS7.FieldByName('VS
2951    Inv').AsFloat;
2952                                end;
2953
2954                  //RFPC0018 poles
2955                  if (RCLLI <> dm_ICM.t_SS7_Location.FieldByName('DB800_PriSCP1').AsString) and
2956                     (RCLLI <> dm_ICM.t_SS7_Location.FieldByName('DB800_PriSCP2').AsString)
```

```
2957                    then if GetSS7Temp(dm_ICM.t_SS7_Location.FieldByName('DB800_PriSCP1').AsString,
2958    'STPPORTPOLE')
2959                            then begin
2960                                  DB800PORTPOLE[1] := DB800PORTPOLE[1] + t_T1_SS7.FieldByName('Total
2961    Inv').AsFloat;
2962                                  DB800PORTPOLE[2] := DB800PORTPOLE[2] + t_T1_SS7.FieldByName('VS
2963    Inv').AsFloat;
2964                                end;
2965                  if (RCLLI <> dm_ICM.t_SS7_Location.FieldByName('DB800_SecSCP1').AsString) and
2966                     (RCLLI <> dm_ICM.t_SS7_Location.FieldByName('DB800_SecSCP2').AsString)
2967                    then if GetSS7Temp(dm_ICM.t_SS7_Location.FieldByName('DB800_SecSCP1').AsString,
2968    'STPPORTPOLE')
2969                            then begin
2970                                  DB800PORTPOLE[1] := DB800PORTPOLE[1] + t_T1_SS7.FieldByName('Total
2971    Inv').AsFloat;
2972                                  DB800PORTPOLE[2] := DB800PORTPOLE[2] + t_T1_SS7.FieldByName('VS
2973    Inv').AsFloat;
2974                                end;
2975
2976                  //RFPC0018 conduit
2977                  if (RCLLI <> dm_ICM.t_SS7_Location.FieldByName('DB800_PriSCP1').AsString) and
2978                     (RCLLI <> dm_ICM.t_SS7_Location.FieldByName('DB800_PriSCP2').AsString)
2979                    then if GetSS7Temp(dm_ICM.t_SS7_Location.FieldByName('DB800_PriSCP1').AsString,
2980    'STPPORTCOND')
2981                            then begin
2982                                  DB800PORTCOND[1] := DB800PORTCOND[1] + t_T1_SS7.FieldByName('Total
2983    Inv').AsFloat;
2984                                  DB800PORTCOND[2] := DB800PORTCOND[2] + t_T1_SS7.FieldByName('VS
2985    Inv').AsFloat;
2986                                end;
2987                  if (RCLLI <> dm_ICM.t_SS7_Location.FieldByName('DB800_SecSCP1').AsString) and
2988                     (RCLLI <> dm_ICM.t_SS7_Location.FieldByName('DB800_SecSCP2').AsString)
2989                    then if GetSS7Temp(dm_ICM.t_SS7_Location.FieldByName('DB800_SecSCP1').AsString,
2990    'STPPORTCOND')
2991                            then begin
2992                                  DB800PORTCOND[1] := DB800PORTCOND[1] + t_T1_SS7.FieldByName('Total
2993    Inv').AsFloat;
2994                                  DB800PORTCOND[2] := DB800PORTCOND[2] + t_T1_SS7.FieldByName('VS
2995    Inv').AsFloat;
```

```
2996                              end;
2997
2998                  for I := 1 to 2 do
2999                    begin
3000                      DB800PORTSWT[I] := DB800PORTSWT[I];
3001                      DB800PORTCKT[I] := DB800PORTCKT[I];
3002                      DB800PORTAERF[I] := DB800PORTAERF[I];
3003                      DB800PORTBURF[I] := DB800PORTBURF[I];
3004                      DB800PORTUNDF[I] := DB800PORTUNDF[I];
3005                      DB800PORTPOLE[I] := DB800PORTPOLE[I];
3006                      DB800PORTCOND[I] := DB800PORTCOND[I];
3007                    end;
3008                end;
3009    //RFPC0018 Section VII Part 10 end
3010
3011    //RFPC0018 Section VII Part 11 begin
3012    {
3013         Each STP requires an investment for the ports at the regional STPs
3014         colocated with LIDB SCPs, which is the average of the port investments
3015         at the primary and secondary pairs of regional STPs for LIDB query
3016         service.  For the variables X below, Xpi refers to the port investment
3017         value stored for primary LIDB STPi (i = 1, 2), and Xsi refers to the port
3018         investment value stored for secondary STPi (i = 1, 2).  The primary LIDB
3019         STPs are identified by the CLLI Codes Reg_Location.LIDB_PriSCP1 and
3020         Reg_Location.LIDB_PriSCP2, and secondary LIDB STPs are identified by the
3021         CLLI Codes Reg_Location.LIDB_SecSCP1 and Reg_Location.LIDB_SecSCP2, where
3022         Reg_Location.RegSTP1 = RSTP1.
3023    }
3024         //RFPC0018 LIDB ports
3025         RCLLI := dm_ICM.t_SS7_STP.FieldByName('Regional_STP1').AsString;
3026         if dm_ICM.t_SS7_Location.FindKey([RCLLI])
3027           then begin
3028                  //RFPC0018 software
3029                  if (RCLLI <> dm_ICM.t_SS7_Location.FieldByName('LIDB_PriSCP1').AsString) and
3030                     (RCLLI <> dm_ICM.t_SS7_Location.FieldByName('LIDB_PriSCP2').AsString)
3031                    then if GetSS7Temp(dm_ICM.t_SS7_Location.FieldByName('LIDB_PriSCP1').AsString,
3032    'STPPORTSWT')
3033                           then begin
```

```
3034                                    LIDBPORTSWT[1] := LIDBPORTSWT[1] + t_T1_SS7.FieldByName('Total
3035  Inv').AsFloat;
3036                                    LIDBPORTSWT[2] := LIDBPORTSWT[2] + t_T1_SS7.FieldByName('VS Inv').AsFloat;
3037                                  end;
3038                        if (RCLLI <> dm_ICM.t_SS7_Location.FieldByName('LIDB_SecSCP1').AsString) and
3039                           (RCLLI <> dm_ICM.t_SS7_Location.FieldByName('LIDB_SecSCP2').AsString)
3040                          then if GetSS7Temp(dm_ICM.t_SS7_Location.FieldByName('LIDB_SecSCP1').AsString,
3041  'STPPORTSWT')
3042                                  then begin
3043                                    LIDBPORTSWT[1] := LIDBPORTSWT[1] + t_T1_SS7.FieldByName('Total
3044  Inv').AsFloat;
3045                                    LIDBPORTSWT[2] := LIDBPORTSWT[2] + t_T1_SS7.FieldByName('VS Inv').AsFloat;
3046                                  end;
3047
3048                        //RFPC0018 circuit
3049                        if (RCLLI <> dm_ICM.t_SS7_Location.FieldByName('LIDB_PriSCP1').AsString) and
3050                           (RCLLI <> dm_ICM.t_SS7_Location.FieldByName('LIDB_PriSCP2').AsString)
3051                          then if GetSS7Temp(dm_ICM.t_SS7_Location.FieldByName('LIDB_PriSCP1').AsString,
3052  'STPPORTCKT')
3053                                  then begin
3054                                    LIDBPORTCKT[1] := LIDBPORTCKT[1] + t_T1_SS7.FieldByName('Total
3055  Inv').AsFloat;
3056                                    LIDBPORTCKT[2] := LIDBPORTCKT[2] + t_T1_SS7.FieldByName('VS Inv').AsFloat;
3057                                  end;
3058                        if (RCLLI <> dm_ICM.t_SS7_Location.FieldByName('LIDB_SecSCP1').AsString) and
3059                           (RCLLI <> dm_ICM.t_SS7_Location.FieldByName('LIDB_SecSCP2').AsString)
3060                          then if GetSS7Temp(dm_ICM.t_SS7_Location.FieldByName('LIDB_SecSCP1').AsString,
3061  'STPPORTCKT')
3062                                  then begin
3063                                    LIDBPORTCKT[1] := LIDBPORTCKT[1] + t_T1_SS7.FieldByName('Total
3064  Inv').AsFloat;
3065                                    LIDBPORTCKT[2] := LIDBPORTCKT[2] + t_T1_SS7.FieldByName('VS Inv').AsFloat;
3066                                  end;
3067
3068                        //RFPC0018 arial cable
3069                        if (RCLLI <> dm_ICM.t_SS7_Location.FieldByName('LIDB_PriSCP1').AsString) and
3070                           (RCLLI <> dm_ICM.t_SS7_Location.FieldByName('LIDB_PriSCP2').AsString)
3071                          then if GetSS7Temp(dm_ICM.t_SS7_Location.FieldByName('LIDB_PriSCP1').AsString,
3072  'STPPORTAERF')
```

```
3073                            then begin
3074                                  LIDBPORTAERF[1] := LIDBPORTAERF[1] + t_T1_SS7.FieldByName('Total
3075    Inv').AsFloat;
3076                                  LIDBPORTAERF[2] := LIDBPORTAERF[2] + t_T1_SS7.FieldByName('VS Inv').AsFloat;
3077                                end;
3078                  if (RCLLI <> dm_ICM.t_SS7_Location.FieldByName('LIDB_SecSCP1').AsString) and
3079                     (RCLLI <> dm_ICM.t_SS7_Location.FieldByName('LIDB_SecSCP2').AsString)
3080                    then if GetSS7Temp(dm_ICM.t_SS7_Location.FieldByName('LIDB_SecSCP1').AsString,
3081    'STPPORTAERF')
3082                            then begin
3083                                  LIDBPORTAERF[1] := LIDBPORTAERF[1] + t_T1_SS7.FieldByName('Total
3084    Inv').AsFloat;
3085                                  LIDBPORTAERF[2] := LIDBPORTAERF[2] + t_T1_SS7.FieldByName('VS Inv').AsFloat;
3086                                end;
3087
3088                  //RFPC0018 buried cable
3089                  if (RCLLI <> dm_ICM.t_SS7_Location.FieldByName('LIDB_PriSCP1').AsString) and
3090                     (RCLLI <> dm_ICM.t_SS7_Location.FieldByName('LIDB_PriSCP2').AsString)
3091                    then if GetSS7Temp(dm_ICM.t_SS7_Location.FieldByName('LIDB_PriSCP1').AsString,
3092    'STPPORTBURF')
3093                            then begin
3094                                  LIDBPORTBURF[1] := LIDBPORTBURF[1] + t_T1_SS7.FieldByName('Total
3095    Inv').AsFloat;
3096                                  LIDBPORTBURF[2] := LIDBPORTBURF[2] + t_T1_SS7.FieldByName('VS Inv').AsFloat;
3097                                end;
3098                  if (RCLLI <> dm_ICM.t_SS7_Location.FieldByName('LIDB_SecSCP1').AsString) and
3099                     (RCLLI <> dm_ICM.t_SS7_Location.FieldByName('LIDB_SecSCP2').AsString)
3100                    then if GetSS7Temp(dm_ICM.t_SS7_Location.FieldByName('LIDB_SecSCP1').AsString,
3101    'STPPORTBURF')
3102                            then begin
3103                                  LIDBPORTBURF[1] := LIDBPORTBURF[1] + t_T1_SS7.FieldByName('Total
3104    Inv').AsFloat;
3105                                  LIDBPORTBURF[2] := LIDBPORTBURF[2] + t_T1_SS7.FieldByName('VS Inv').AsFloat;
3106                                end;
3107
3108                  //RFPC0018 underground cable
3109                  if (RCLLI <> dm_ICM.t_SS7_Location.FieldByName('LIDB_PriSCP1').AsString) and
3110                     (RCLLI <> dm_ICM.t_SS7_Location.FieldByName('LIDB_PriSCP2').AsString)
```

```
3111                    then if GetSS7Temp(dm_ICM.t_SS7_Location.FieldByName('LIDB_PriSCP1').AsString,
3112  'STPPORTUNDF')
3113                          then begin
3114                                LIDBPORTUNDF[1] := LIDBPORTUNDF[1] + t_T1_SS7.FieldByName('Total
3115  Inv').AsFloat;
3116                                LIDBPORTUNDF[2] := LIDBPORTUNDF[2] + t_T1_SS7.FieldByName('VS Inv').AsFloat;
3117                               end;
3118                  if (RCLLI <> dm_ICM.t_SS7_Location.FieldByName('LIDB_SecSCP1').AsString) and
3119                     (RCLLI <> dm_ICM.t_SS7_Location.FieldByName('LIDB_SecSCP2').AsString)
3120                    then if GetSS7Temp(dm_ICM.t_SS7_Location.FieldByName('LIDB_SecSCP1').AsString,
3121  'STPPORTUNDF')
3122                          then begin
3123                                LIDBPORTUNDF[1] := LIDBPORTUNDF[1] + t_T1_SS7.FieldByName('Total
3124  Inv').AsFloat;
3125                                LIDBPORTUNDF[2] := LIDBPORTUNDF[2] + t_T1_SS7.FieldByName('VS Inv').AsFloat;
3126                               end;
3127
3128                  //RFPC0018 poles
3129                  if (RCLLI <> dm_ICM.t_SS7_Location.FieldByName('LIDB_PriSCP1').AsString) and
3130                     (RCLLI <> dm_ICM.t_SS7_Location.FieldByName('LIDB_PriSCP2').AsString)
3131                    then if GetSS7Temp(dm_ICM.t_SS7_Location.FieldByName('LIDB_PriSCP1').AsString,
3132  'STPPORTPOLE')
3133                          then begin
3134                                LIDBPORTPOLE[1] := LIDBPORTPOLE[1] + t_T1_SS7.FieldByName('Total
3135  Inv').AsFloat;
3136                                LIDBPORTPOLE[2] := LIDBPORTPOLE[2] + t_T1_SS7.FieldByName('VS Inv').AsFloat;
3137                               end;
3138                  if (RCLLI <> dm_ICM.t_SS7_Location.FieldByName('LIDB_SecSCP1').AsString) and
3139                     (RCLLI <> dm_ICM.t_SS7_Location.FieldByName('LIDB_SecSCP2').AsString)
3140                    then if GetSS7Temp(dm_ICM.t_SS7_Location.FieldByName('LIDB_SecSCP1').AsString,
3141  'STPPORTPOLE')
3142                          then begin
3143                                LIDBPORTPOLE[1] := LIDBPORTPOLE[1] + t_T1_SS7.FieldByName('Total
3144  Inv').AsFloat;
3145                                LIDBPORTPOLE[2] := LIDBPORTPOLE[2] + t_T1_SS7.FieldByName('VS Inv').AsFloat;
3146                               end;
3147
3148                  //RFPC0018 conduit
3149                  if (RCLLI <> dm_ICM.t_SS7_Location.FieldByName('LIDB_PriSCP1').AsString) and
```

```
3150                    (RCLLI <> dm_ICM.t_SS7_Location.FieldByName('LIDB_PriSCP2').AsString)
3151                  then if GetSS7Temp(dm_ICM.t_SS7_Location.FieldByName('LIDB_PriSCP1').AsString,
3152  'STPPORTCOND')
3153                         then begin
3154                                LIDBPORTCOND[1] := LIDBPORTCOND[1] + t_T1_SS7.FieldByName('Total
3155  Inv').AsFloat;
3156                                LIDBPORTCOND[2] := LIDBPORTCOND[2] + t_T1_SS7.FieldByName('VS Inv').AsFloat;
3157                              end;
3158                if (RCLLI <> dm_ICM.t_SS7_Location.FieldByName('LIDB_SecSCP1').AsString) and
3159                   (RCLLI <> dm_ICM.t_SS7_Location.FieldByName('LIDB_SecSCP2').AsString)
3160                  then if GetSS7Temp(dm_ICM.t_SS7_Location.FieldByName('LIDB_SecSCP1').AsString,
3161  'STPPORTCOND')
3162                         then begin
3163                                LIDBPORTCOND[1] := LIDBPORTCOND[1] + t_T1_SS7.FieldByName('Total
3164  Inv').AsFloat;
3165                                LIDBPORTCOND[2] := LIDBPORTCOND[2] + t_T1_SS7.FieldByName('VS Inv').AsFloat;
3166                              end;
3167
3168                for I := 1 to 2 do
3169                  begin
3170                    LIDBPORTSWT[I] := LIDBPORTSWT[I];
3171                    LIDBPORTCKT[I] := LIDBPORTCKT[I];
3172                    LIDBPORTAERF[I] := LIDBPORTAERF[I];
3173                    LIDBPORTBURF[I] := LIDBPORTBURF[I];
3174                    LIDBPORTUNDF[I] := LIDBPORTUNDF[I];
3175                    LIDBPORTPOLE[I] := LIDBPORTPOLE[I];
3176                    LIDBPORTCOND[I] := LIDBPORTCOND[I];
3177                  end;
3178              end;
3179   //RFPC0018 Section VII Part 11 end
3180
3181        //RFPC0018 write data out to a temporary table
3182        WriteTemp(CLLI, 'DLINKCKT',     DLINKCKT );
3183        WriteTemp(CLLI, 'DLINKAERF',    DLINKAERF);
3184        WriteTemp(CLLI, 'DLINKBURF',    DLINKBURF);
3185        WriteTemp(CLLI, 'DLINKUNDF',    DLINKUNDF);
3186        WriteTemp(CLLI, 'DLINKPOLE',    DLINKPOLE);
3187        WriteTemp(CLLI, 'DLINKCOND',    DLINKCOND);
3188        WriteTemp(CLLI, 'DLINKEXP',     DLINKEXP );
```

```
3189        WriteTemp(CLLI, 'DB800LINKCKT', DB800LINKCKT );
3190        WriteTemp(CLLI, 'LIDBLINKCKT',  LIDBLINKCKT );
3191        WriteTemp(CLLI, 'DB800LINKEXP', DB800LINKEXP );
3192        WriteTemp(CLLI, 'LIDBLINKEXP',  LIDBLINKEXP );
3193        WriteTemp(CLLI, 'ALINK800CKT',  ALINK800CKT );
3194        WriteTemp(CLLI, 'ALINK800AERF', ALINK800AERF );
3195        WriteTemp(CLLI, 'ALINK800BURF', ALINK800BURF );
3196        WriteTemp(CLLI, 'ALINK800UNDF', ALINK800UNDF );
3197        WriteTemp(CLLI, 'ALINK800POLE', ALINK800POLE );
3198        WriteTemp(CLLI, 'ALINK800COND', ALINK800COND );
3199        WriteTemp(CLLI, 'ALINKLIDBCKT', ALINKLIDBCKT );
3200        WriteTemp(CLLI, 'ALINKLIDBAERF',ALINKLIDBAERF );
3201        WriteTemp(CLLI, 'ALINKLIDBBURF',ALINKLIDBBURF );
3202        WriteTemp(CLLI, 'ALINKLIDBUNDF',ALINKLIDBUNDF );
3203        WriteTemp(CLLI, 'ALINKLIDBPOLE',ALINKLIDBPOLE );
3204        WriteTemp(CLLI, 'ALINKLIDBCOND',ALINKLIDBCOND );
3205        WriteTemp(CLLI, 'GTTInv_R',     GTTInv_R);
3206        WriteTemp(CLLI, 'TCAPSWR',      TCAPSWR);
3207        WriteTemp(CLLI, 'RSPORTSWT',    RSPORTSWT);
3208        WriteTemp(CLLI, 'RSPORTCKT',    RSPORTCKT);
3209        WriteTemp(CLLI, 'RSPORTAERF',   RSPORTAERF);
3210        WriteTemp(CLLI, 'RSPORTBURF',   RSPORTBURF);
3211        WriteTemp(CLLI, 'RSPORTUNDF',   RSPORTUNDF);
3212        WriteTemp(CLLI, 'RSPORTPOLE',   RSPORTPOLE);
3213        WriteTemp(CLLI, 'RSPORTCOND',   RSPORTCOND);
3214        WriteTemp(CLLI, 'DB800PORTSWT', DB800PORTSWT);
3215        WriteTemp(CLLI, 'DB800PORTCKT', DB800PORTCKT);
3216        WriteTemp(CLLI, 'DB800PORTAERF',DB800PORTAERF);
3217        WriteTemp(CLLI, 'DB800PORTBURF',DB800PORTBURF);
3218        WriteTemp(CLLI, 'DB800PORTUNDF',DB800PORTUNDF);
3219        WriteTemp(CLLI, 'DB800PORTPOLE',DB800PORTPOLE);
3220        WriteTemp(CLLI, 'DB800PORTCOND',DB800PORTCOND);
3221        WriteTemp(CLLI, 'LIDBPORTSWT',  LIDBPORTSWT);
3222        WriteTemp(CLLI, 'LIDBPORTCKT',  LIDBPORTCKT);
3223        WriteTemp(CLLI, 'LIDBPORTAERF', LIDBPORTAERF);
3224        WriteTemp(CLLI, 'LIDBPORTBURF', LIDBPORTBURF);
3225        WriteTemp(CLLI, 'LIDBPORTUNDF', LIDBPORTUNDF);
3226        WriteTemp(CLLI, 'LIDBPORTPOLE', LIDBPORTPOLE);
3227        WriteTemp(CLLI, 'LIDBPORTCOND', LIDBPORTCOND);
```

```
3228
3229          dm_ICM.t_SS7_STP.Next;
3230        end;
3231
3232    end;
3233
3234    procedure Tf_ss7.SS7_Invst_SCP;
3235    Var
3236      CLLI       : String;
3237      Inv        : TInvestment;
3238      I          : Integer;
3239      X          : Double;
3240
3241     // This procedure produces SCP investment data. The information is sent to the
3242     // temporary table.
3243    begin
3244      //RFPC0018 Section VII Part 14 Begin
3245     {
3246         14.  SCP Investments per Query
3247
3248              The total investment value SCPInv is converted to an investment per
3249              query.
3250
3251              Variables used from the Network Parameters Table:
3252
3253                 SCPALinks = Number of A Links between STP and SCP
3254                 OctetCap = Octet Capacity per Link
3255                 DB800Octets = Octets per DB800 Query
3256                 LIDBOctets = Octets per LIDB Query
3257                 DB800Usage = 800/888 SCP Average Utilization
3258                 LIDBUsage = LIDB SCP Average Utilization
3259
3260              Now calculate investments per query.
3261
3262              Volume-Sensitive Investments:
3263
3264                SCP800 = SCPInv / (SCPALinks * OctetCap / DB800Octets)
3265                SCPLIDB = SCPInv / (SCPALinks * OctetCap / LIDBOctets)
3266
```

```
3267            Average Investments:
3268
3269              SCP800 = SCPInv / (SCPALinks * OctetCap * DB800Usage / DB800Octets)
3270              SCPLIDB = SCPInv / (SCPALinks * OctetCap * LIDBUsage / LIDBOctets)
3271
3272    }
3273     CLLI := 'SCP0XX00000';
3274
3275     dm_ICM.t_SS7_Investment.CancelRange; //RFPC0018 Clear filter on SS7 investment table
3276     dm_ICM.t_SS7_Investment.SetRange(['SCP'], ['SCP']) ; //RFPC0018 Set filter on SS7 investment table
3277     dm_ICM.t_SS7_Investment.First;
3278     while Not dm_ICM.t_SS7_Investment.EOF do
3279       begin
3280         Application.ProcessMessages;
3281         Inv[1] := (dm_ICM.t_SS7_Investment.FieldByName('Hardware').AsFloat *
3282                 (1.0 + dm_ICM.t_SS7_Loading.FieldByName('HW_Factor').AsFloat )) +
3283                (dm_ICM.t_SS7_Investment.FieldByName('Software').AsFloat *
3284                 (1.0 + dm_ICM.t_SS7_Loading.FieldByName('SW_Factor').AsFloat )) +
3285                (dm_ICM.t_SS7_Investment.FieldByName('Eng_Hrs').AsFloat *
3286                 dm_ICM.t_SS7_Loading.FieldByName('Eng_Rate').AsFloat ) +
3287                (dm_ICM.t_SS7_Investment.FieldByName('Inst_Hrs').AsFloat *
3288                 dm_ICM.t_SS7_Loading.FieldByName('Inst_Rate').AsFloat ) +
3289                (dm_ICM.t_SS7_Investment.FieldByName('Test_Hrs').AsFloat *
3290                 dm_ICM.t_SS7_Loading.FieldByName('Test_Rate').AsFloat );
3291         Inv[2] := Inv[1];
3292         WriteTemp(CLLI, dm_ICM.t_SS7_Investment.FieldByName('Item').AsString,  Inv);
3293
3294         dm_ICM.t_SS7_Investment.Next;
3295       end;
3296
3297     if GetSS7Temp(CLLI, 'SCPInv')
3298       then begin
3299             //RFPC0018 database 800
3300             Inv[1] := t_T1_SS7.FieldByName('Total Inv').AsFloat /
3301                       (GetParameter('SCPALinks') * GetParameter('OctetCap') *
3302                        GetParameter('DB800Util') / GetParameter('DB800Octets') );
3303             Inv[2] := t_T1_SS7.FieldByName('VS Inv').AsFloat /
3304                       (GetParameter('SCPALinks') * GetParameter('OctetCap') /
3305                        GetParameter('DB800Octets') );
```

```
3306              WriteTemp(CLLI, 'SCP800',  Inv);
3307            end;
3308
3309    if GetSS7Temp(CLLI, 'SCPInv')
3310      then begin
3311              //RFPC0018 LIDB
3312              Inv[1] := t_T1_SS7.FieldByName('Total Inv').AsFloat /
3313                        (GetParameter('SCPALinks') * GetParameter('OctetCap') *
3314                         GetParameter('LIDBUtil') / GetParameter('LIDBOctets') );
3315              Inv[2] := t_T1_SS7.FieldByName('VS Inv').AsFloat /
3316                        (GetParameter('SCPALinks') * GetParameter('OctetCap') /
3317                         GetParameter('LIDBOctets') );
3318              WriteTemp(CLLI, 'SCPLIDB',  Inv);
3319            end;
3320
3321    if GetSS7Temp(CLLI, 'SCPInv')
3322      then begin
3323              //RFPC0018 LNP
3324              Inv[1] := t_T1_SS7.FieldByName('Total Inv').AsFloat /
3325                        (GetParameter('SCPALinks') * GetParameter('OctetCap') *
3326                         GetParameter('LNPUtil') / GetParameter('LNPOctets') );
3327              Inv[2] := t_T1_SS7.FieldByName('VS Inv').AsFloat /
3328                        (GetParameter('SCPALinks') * GetParameter('OctetCap') /
3329                         GetParameter('LNPOctets') );
3330              WriteTemp(CLLI, 'SCPLNP',  Inv);
3331            end;
3332    //RFPC0018 Section VII Part 14 end
3333  end;
3334
3335  procedure Tf_ss7.OutputHostInvestments;
3336   var
3337     Inv  : TInvestment;
3338     CLLI : String;
3339     I    : integer;
3340
3341   // This procedure produces the ACTSS7.DB table. It pulls data from the
3342   // temporary table to create the investment componements for SS7.
3343   begin
3344     dm_ICM.t_SS7_SSP.First;          //RFPC0018 Go to the beginning of the SSP table
```

```
3345      while Not dm_ICM.t_SS7_SSP.EOF do
3346        begin
3347          Application.ProcessMessages;
3348          CLLI := dm_ICM.t_SS7_SSP.FieldByName('SSP_CLLI').AsString;
3349          if dm_ICM.t_Nodes.FindKey([CLLI]) //RFPC0018 Find CLLI code in node table
3350            then begin  //RFPC0018 CLLI code found, output data
3351
3352                  //RFPC0018 Octets per SSP - STP A Link
3353                  Inv[1] := GetParameter('OctetCap') * GetParameter('ALinkUtil');
3354                  Inv[2] := Inv[1];
3355                  Inv[3] := Inv[1];
3356                  WriteSS7Act(dm_ICM.t_A_SS7H, CLLI, 'ALINKOCTETS', '000000', Inv);
3357
3358                  //RFPC0018 Octets per local - Regional STP D Link
3359                  Inv[1] := GetParameter('OctetCap') * GetParameter('DLinkUtil');
3360                  Inv[2] := Inv[1];
3361                  Inv[3] := Inv[1];
3362                  WriteSS7Act(dm_ICM.t_A_SS7H, CLLI, 'DLINKOCTETS', '000000', Inv);
3363
3364                  //RFPC0018 Octets per D Link to LIDB Reg STP
3365                  Inv[1] := GetParameter('OctetCap') * GetParameter('LIDBUtil');
3366                  Inv[2] := Inv[1];
3367                  Inv[3] := Inv[1];
3368                  WriteSS7Act(dm_ICM.t_A_SS7H, CLLI, 'LINKLIDBOCTETS', '000000', Inv);
3369
3370                  //RFPC0018 Octets per D Link to 800 Reg STP
3371                  Inv[1] := GetParameter('OctetCap') * GetParameter('DB800Util');
3372                  Inv[2] := Inv[1];
3373                  Inv[3] := Inv[1];
3374                  WriteSS7Act(dm_ICM.t_A_SS7H, CLLI, 'LINK800OCTETS', '000000', Inv);
3375
3376                  //RFPC0018 Call Octets
3377                  Inv[1] := GetParameter('CallOctets');
3378                  Inv[2] := Inv[1];
3379                  Inv[3] := Inv[1];
3380                  WriteSS7Act(dm_ICM.t_A_SS7H, CLLI, 'CALLOCTETS', '000000', Inv);
3381
3382                  //RFPC0018 DB 800  Octets
3383                  Inv[1] := GetParameter('DB800Octets');
```

```
3384                Inv[2] := Inv[1];
3385                Inv[3] := Inv[1];
3386                WriteSS7Act(dm_ICM.t_A_SS7H, CLLI, 'DB800OCTETS', '000000', Inv);
3387
3388                //RFPC0018 LIDB Octets
3389                Inv[1] := GetParameter('LIDBOctets');
3390                Inv[2] := Inv[1];
3391                Inv[3] := Inv[1];
3392                WriteSS7Act(dm_ICM.t_A_SS7H, CLLI, 'LIDBOCTETS', '000000', Inv);
3393
3394                //RFPC0018 LNP Octets
3395                Inv[1] := GetParameter('LNPOctets');
3396                Inv[2] := Inv[1];
3397                Inv[3] := Inv[1];
3398                WriteSS7Act(dm_ICM.t_A_SS7H, CLLI, 'LNPOCTETS', '000000', Inv);
3399
3400               //RFPC0018 A Links
3401               if GetSS7Temp(CLLI, 'ALINKS')
3402                  then begin
3403                         Inv[1] := t_T1_SS7.FieldByName('Total Inv').AsFloat;
3404                         Inv[2] := Inv[1];
3405                         Inv[3] := Inv[1];
3406                       end
3407                  else begin
3408                         Inv[1] := 0;
3409                         Inv[2] := 0;
3410                         Inv[3] := 0;
3411                       end;
3412                WriteSS7Act(dm_ICM.t_A_SS7H, CLLI, 'ALINKS', '000000', Inv);
3413
3414                //RFPC0018 D Link to 800 Reg STP - Circuit Equip
3415                if GetSS7Temp(dm_ICM.t_SS7_SSP.FieldByName('Local_STP1').AsString, 'DB800LINKCKT')
3416                  then begin
3417                         Inv[1] := t_T1_SS7.FieldByName('Total Inv').AsFloat;
3418                         Inv[2] := t_T1_SS7.FieldByName('VS Inv').AsFloat;
3419                       end
3420                  else begin
3421                         Inv[1] := 0;
3422                         Inv[2] := 0;
```

```
3423                    end;
3424                  if GetSS7Temp(dm_ICM.t_SS7_SSP.FieldByName('Local_STP2').AsString, 'DB800LINKCKT')
3425                    then begin
3426                           Inv[1] := Inv[1] + t_T1_SS7.FieldByName('Total Inv').AsFloat;
3427                           Inv[2] := Inv[2] + t_T1_SS7.FieldByName('VS Inv').AsFloat;
3428                         end;
3429                  Inv[3] := Inv[1] - Inv[2];
3430                  for I := 1 to 3 do Inv[I] := Inv[I] / 2.0;
3431                  WriteSS7Act(dm_ICM.t_A_SS7H, CLLI, 'DLINK800CKT', '223200', Inv);
3432
3433                  //RFPC0018 D Link to 800 Reg STP - Lease Expense
3434                  if GetSS7Temp(dm_ICM.t_SS7_SSP.FieldByName('Local_STP1').AsString, 'DB800LINKEXP')
3435                    then begin
3436                           Inv[1] := t_T1_SS7.FieldByName('Total Inv').AsFloat;
3437                           Inv[2] := t_T1_SS7.FieldByName('VS Inv').AsFloat;
3438                         end
3439                    else begin
3440                           Inv[1] := 0;
3441                           Inv[2] := 0;
3442                         end;
3443                  if GetSS7Temp(dm_ICM.t_SS7_SSP.FieldByName('Local_STP2').AsString, 'DB800LINKEXP')
3444                    then begin
3445                           Inv[1] := Inv[1] + t_T1_SS7.FieldByName('Total Inv').AsFloat;
3446                           Inv[2] := Inv[2] + t_T1_SS7.FieldByName('VS Inv').AsFloat;
3447                         end;
3448                  Inv[3] := Inv[1] - Inv[2];
3449                  for I := 1 to 3 do Inv[I] := Inv[I] / 2.0;
3450                  WriteSS7Act(dm_ICM.t_A_SS7H, CLLI, 'DLINK800EXP', '621214', Inv);
3451
3452                 //RFPC0018 Port at 800 Reg STP
3453                  if GetSS7Temp(dm_ICM.t_SS7_SSP.FieldByName('Local_STP1').AsString, 'DB800PORTSWT')
3454                    then begin
3455                           Inv[1] := t_T1_SS7.FieldByName('Total Inv').AsFloat;
3456                           Inv[2] := t_T1_SS7.FieldByName('VS Inv').AsFloat;
3457                         end
3458                    else begin
3459                           Inv[1] := 0;
3460                           Inv[2] := 0;
3461                         end;
```

```
3462                if GetSS7Temp(dm_ICM.t_SS7_SSP.FieldByName('Local_STP2').AsString, 'DB800PORTSWT')
3463                  then begin
3464                         Inv[1] := Inv[1] + t_T1_SS7.FieldByName('Total Inv').AsFloat;
3465                         Inv[2] := Inv[2] + t_T1_SS7.FieldByName('VS Inv').AsFloat;
3466                       end;
3467                Inv[3] := Inv[1] - Inv[2];
3468                for I := 1 to 3 do Inv[I] := Inv[I] / 2.0;
3469                WriteSS7Act(dm_ICM.t_A_SS7H, CLLI, 'PORT800SWT', '223200', Inv);
3470
3471                //RFPC0018 Port at 800 Reg STP - Aerial Fiber
3472                if GetSS7Temp(dm_ICM.t_SS7_SSP.FieldByName('Local_STP1').AsString, 'DB800PORTAERF')
3473                  then begin
3474                         Inv[1] := t_T1_SS7.FieldByName('Total Inv').AsFloat;
3475                         Inv[2] := t_T1_SS7.FieldByName('VS Inv').AsFloat;
3476                       end
3477                  else begin
3478                         Inv[1] := 0;
3479                         Inv[2] := 0;
3480                       end;
3481                if GetSS7Temp(dm_ICM.t_SS7_SSP.FieldByName('Local_STP2').AsString, 'DB800PORTAERF')
3482                  then begin
3483                         Inv[1] := Inv[1] + t_T1_SS7.FieldByName('Total Inv').AsFloat;
3484                         Inv[2] := Inv[2] + t_T1_SS7.FieldByName('VS Inv').AsFloat;
3485                       end;
3486                Inv[3] := Inv[1] - Inv[2];
3487                for I := 1 to 3 do Inv[I] := Inv[I] / 2.0;
3488                WriteSS7Act(dm_ICM.t_A_SS7H, CLLI, 'PORT800AERF', '242100', Inv);
3489
3490                //RFPC0018 Port at 800 Reg STP - Buried Fiber
3491                if GetSS7Temp(dm_ICM.t_SS7_SSP.FieldByName('Local_STP1').AsString, 'DB800PORTBURF')
3492                  then begin
3493                         Inv[1] := t_T1_SS7.FieldByName('Total Inv').AsFloat;
3494                         Inv[2] := t_T1_SS7.FieldByName('VS Inv').AsFloat;
3495                       end
3496                  else begin
3497                         Inv[1] := 0;
3498                         Inv[2] := 0;
3499                       end;
3500                if GetSS7Temp(dm_ICM.t_SS7_SSP.FieldByName('Local_STP2').AsString, 'DB800PORTBURF')
```

```
3501                    then begin
3502                          Inv[1] := Inv[1] + t_T1_SS7.FieldByName('Total Inv').AsFloat;
3503                          Inv[2] := Inv[2] + t_T1_SS7.FieldByName('VS Inv').AsFloat;
3504                        end;
3505                  Inv[3] := Inv[1] - Inv[2];
3506                  for I := 1 to 3 do Inv[I] := Inv[I] / 2.0;
3507                  WriteSS7Act(dm_ICM.t_A_SS7H, CLLI, 'PORT800BURF', '242300', Inv);
3508
3509                  //RFPC0018 Port at 800 Reg STP - Underground Fiber
3510                  if GetSS7Temp(dm_ICM.t_SS7_SSP.FieldByName('Local_STP1').AsString, 'DB800PORTUNDF')
3511                    then begin
3512                          Inv[1] := t_T1_SS7.FieldByName('Total Inv').AsFloat;
3513                          Inv[2] := t_T1_SS7.FieldByName('VS Inv').AsFloat;
3514                        end
3515                    else begin
3516                          Inv[1] := 0;
3517                          Inv[2] := 0;
3518                        end;
3519                  if GetSS7Temp(dm_ICM.t_SS7_SSP.FieldByName('Local_STP2').AsString, 'DB800PORTUNDF')
3520                    then begin
3521                          Inv[1] := Inv[1] + t_T1_SS7.FieldByName('Total Inv').AsFloat;
3522                          Inv[2] := Inv[2] + t_T1_SS7.FieldByName('VS Inv').AsFloat;
3523                        end;
3524                  Inv[3] := Inv[1] - Inv[2];
3525                  for I := 1 to 3 do Inv[I] := Inv[I] / 2.0;
3526                  WriteSS7Act(dm_ICM.t_A_SS7H, CLLI, 'PORT800UNDF', '242220', Inv);
3527
3528                  //RFPC0018 Port at 800 Reg STP - Circuit equip
3529                  if GetSS7Temp(dm_ICM.t_SS7_SSP.FieldByName('Local_STP1').AsString, 'DB800PORTCKT')
3530                    then begin
3531                          Inv[1] := t_T1_SS7.FieldByName('Total Inv').AsFloat;
3532                          Inv[2] := t_T1_SS7.FieldByName('VS Inv').AsFloat;
3533                        end
3534                    else begin
3535                          Inv[1] := 0;
3536                          Inv[2] := 0;
3537                        end;
3538                  if GetSS7Temp(dm_ICM.t_SS7_SSP.FieldByName('Local_STP2').AsString, 'DB800PORTCKT')
3539                    then begin
```

```
3540                    Inv[1] := Inv[1] + t_T1_SS7.FieldByName('Total Inv').AsFloat;
3541                    Inv[2] := Inv[2] + t_T1_SS7.FieldByName('VS Inv').AsFloat;
3542                  end;
3543          Inv[3] := Inv[1] - Inv[2];
3544          for I := 1 to 3 do Inv[I] := Inv[I] / 2.0;
3545          WriteSS7Act(dm_ICM.t_A_SS7H, CLLI, 'PORT800CKT', '223200', Inv);
3546
3547          //RFPC0018 Port at 800 Reg STP - Conduit
3548          if GetSS7Temp(dm_ICM.t_SS7_SSP.FieldByName('Local_STP1').AsString, 'DB800PORTCOND')
3549            then begin
3550                    Inv[1] := t_T1_SS7.FieldByName('Total Inv').AsFloat;
3551                    Inv[2] := t_T1_SS7.FieldByName('VS Inv').AsFloat;
3552                  end
3553            else begin
3554                    Inv[1] := 0;
3555                    Inv[2] := 0;
3556                  end;
3557          if GetSS7Temp(dm_ICM.t_SS7_SSP.FieldByName('Local_STP2').AsString, 'DB800PORTCOND')
3558            then begin
3559                    Inv[1] := Inv[1] + t_T1_SS7.FieldByName('Total Inv').AsFloat;
3560                    Inv[2] := Inv[2] + t_T1_SS7.FieldByName('VS Inv').AsFloat;
3561                  end;
3562          Inv[3] := Inv[1] - Inv[2];
3563          for I := 1 to 3 do Inv[I] := Inv[I] / 2.0;
3564          WriteSS7Act(dm_ICM.t_A_SS7H, CLLI, 'PORT800COND', '244100', Inv);
3565
3566          //RFPC0018 Port at 800 Reg STP - Poles
3567          if GetSS7Temp(dm_ICM.t_SS7_SSP.FieldByName('Local_STP1').AsString, 'DB800PORTPOLE')
3568            then begin
3569                    Inv[1] := t_T1_SS7.FieldByName('Total Inv').AsFloat;
3570                    Inv[2] := t_T1_SS7.FieldByName('VS Inv').AsFloat;
3571                  end
3572            else begin
3573                    Inv[1] := 0;
3574                    Inv[2] := 0;
3575                  end;
3576          if GetSS7Temp(dm_ICM.t_SS7_SSP.FieldByName('Local_STP2').AsString, 'DB800PORTPOLE')
3577            then begin
3578                    Inv[1] := Inv[1] + t_T1_SS7.FieldByName('Total Inv').AsFloat;
```

```
3579                    Inv[2] := Inv[2] + t_T1_SS7.FieldByName('VS Inv').AsFloat;
3580                  end;
3581          Inv[3] := Inv[1] - Inv[2];
3582          for I := 1 to 3 do Inv[I] := Inv[I] / 2.0;
3583          WriteSS7Act(dm_ICM.t_A_SS7H, CLLI, 'PORT800POLE', '241100', Inv);
3584
3585          //RFPC0018 Port at LIDB Reg STP
3586          if GetSS7Temp(dm_ICM.t_SS7_SSP.FieldByName('Local_STP1').AsString, 'LIDBPORTSWT')
3587            then begin
3588                    Inv[1] := t_T1_SS7.FieldByName('Total Inv').AsFloat;
3589                    Inv[2] := t_T1_SS7.FieldByName('VS Inv').AsFloat;
3590                  end
3591            else begin
3592                    Inv[1] := 0;
3593                    Inv[2] := 0;
3594                  end;
3595          if GetSS7Temp(dm_ICM.t_SS7_SSP.FieldByName('Local_STP2').AsString, 'LIDBPORTSWT')
3596            then begin
3597                    Inv[1] := Inv[1] + t_T1_SS7.FieldByName('Total Inv').AsFloat;
3598                    Inv[2] := Inv[2] + t_T1_SS7.FieldByName('VS Inv').AsFloat;
3599                  end;
3600          Inv[3] := Inv[1] - Inv[2];
3601          for I := 1 to 3 do Inv[I] := Inv[I] / 2.0;
3602          WriteSS7Act(dm_ICM.t_A_SS7H, CLLI, 'PORTLIDBSWT', '223200', Inv);
3603
3604          //RFPC0018 Port at LIDB Reg STP - Aerial Fiber
3605          if GetSS7Temp(dm_ICM.t_SS7_SSP.FieldByName('Local_STP1').AsString, 'LIDBPORTAERF')
3606            then begin
3607                    Inv[1] := t_T1_SS7.FieldByName('Total Inv').AsFloat;
3608                    Inv[2] := t_T1_SS7.FieldByName('VS Inv').AsFloat;
3609                  end
3610            else begin
3611                    Inv[1] := 0;
3612                    Inv[2] := 0;
3613                  end;
3614          if GetSS7Temp(dm_ICM.t_SS7_SSP.FieldByName('Local_STP2').AsString, 'LIDBPORTAERF')
3615            then begin
3616                    Inv[1] := Inv[1] + t_T1_SS7.FieldByName('Total Inv').AsFloat;
3617                    Inv[2] := Inv[2] + t_T1_SS7.FieldByName('VS Inv').AsFloat;
```

```
3618                end;
3619          Inv[3] := Inv[1] - Inv[2];
3620          for I := 1 to 3 do Inv[I] := Inv[I] / 2.0;
3621          WriteSS7Act(dm_ICM.t_A_SS7H, CLLI, 'PORTLIDBAERF', '242100', Inv);
3622
3623          //RFPC0018 Port at LIDB Reg STP - Buried Fiber
3624          if GetSS7Temp(dm_ICM.t_SS7_SSP.FieldByName('Local_STP1').AsString, 'LIDBPORTBURF')
3625            then begin
3626                   Inv[1] := t_T1_SS7.FieldByName('Total Inv').AsFloat;
3627                   Inv[2] := t_T1_SS7.FieldByName('VS Inv').AsFloat;
3628                 end
3629            else begin
3630                   Inv[1] := 0;
3631                   Inv[2] := 0;
3632                 end;
3633          if GetSS7Temp(dm_ICM.t_SS7_SSP.FieldByName('Local_STP2').AsString, 'LIDBPORTBURF')
3634            then begin
3635                   Inv[1] := Inv[1] + t_T1_SS7.FieldByName('Total Inv').AsFloat;
3636                   Inv[2] := Inv[2] + t_T1_SS7.FieldByName('VS Inv').AsFloat;
3637                 end;
3638          Inv[3] := Inv[1] - Inv[2];
3639          for I := 1 to 3 do Inv[I] := Inv[I] / 2.0;
3640          WriteSS7Act(dm_ICM.t_A_SS7H, CLLI, 'PORTLIDBBURF', '242300', Inv);
3641
3642          //RFPC0018 Port at LIDB Reg STP - Underground Fiber
3643          if GetSS7Temp(dm_ICM.t_SS7_SSP.FieldByName('Local_STP1').AsString, 'LIDBPORTUNDF')
3644            then begin
3645                   Inv[1] := t_T1_SS7.FieldByName('Total Inv').AsFloat;
3646                   Inv[2] := t_T1_SS7.FieldByName('VS Inv').AsFloat;
3647                 end
3648            else begin
3649                   Inv[1] := 0;
3650                   Inv[2] := 0;
3651                 end;
3652          if GetSS7Temp(dm_ICM.t_SS7_SSP.FieldByName('Local_STP2').AsString, 'LIDBPORTUNDF')
3653            then begin
3654                   Inv[1] := Inv[1] + t_T1_SS7.FieldByName('Total Inv').AsFloat;
3655                   Inv[2] := Inv[2] + t_T1_SS7.FieldByName('VS Inv').AsFloat;
3656                 end;
```

```
3657                    Inv[3] := Inv[1] - Inv[2];
3658                    for I := 1 to 3 do Inv[I] := Inv[I] / 2.0;
3659                    WriteSS7Act(dm_ICM.t_A_SS7H, CLLI, 'PORTLIDBUNDF', '242220', Inv);
3660
3661                    //RFPC0018 Port at LIDB STP - Circuit equip
3662                    if GetSS7Temp(dm_ICM.t_SS7_SSP.FieldByName('Local_STP1').AsString, 'LIDBPORTCKT')
3663                      then begin
3664                             Inv[1] := t_T1_SS7.FieldByName('Total Inv').AsFloat;
3665                             Inv[2] := t_T1_SS7.FieldByName('VS Inv').AsFloat;
3666                           end
3667                      else begin
3668                             Inv[1] := 0;
3669                             Inv[2] := 0;
3670                           end;
3671                    if GetSS7Temp(dm_ICM.t_SS7_SSP.FieldByName('Local_STP2').AsString, 'LIDBPORTCKT')
3672                      then begin
3673                             Inv[1] := Inv[1] + t_T1_SS7.FieldByName('Total Inv').AsFloat;
3674                             Inv[2] := Inv[2] + t_T1_SS7.FieldByName('VS Inv').AsFloat;
3675                           end;
3676                    Inv[3] := Inv[1] - Inv[2];
3677                    for I := 1 to 3 do Inv[I] := Inv[I] / 2.0;
3678                    WriteSS7Act(dm_ICM.t_A_SS7H, CLLI, 'PORTLIDBCKT', '223200', Inv);
3679
3680                    //RFPC0018 Port at LIDB Reg STP - Conduit
3681                    if GetSS7Temp(dm_ICM.t_SS7_SSP.FieldByName('Local_STP1').AsString, 'LIDBPORTCOND')
3682                      then begin
3683                             Inv[1] := t_T1_SS7.FieldByName('Total Inv').AsFloat;
3684                             Inv[2] := t_T1_SS7.FieldByName('VS Inv').AsFloat;
3685                           end
3686                      else begin
3687                             Inv[1] := 0;
3688                             Inv[2] := 0;
3689                           end;
3690                    if GetSS7Temp(dm_ICM.t_SS7_SSP.FieldByName('Local_STP2').AsString, 'LIDBPORTCOND')
3691                      then begin
3692                             Inv[1] := Inv[1] + t_T1_SS7.FieldByName('Total Inv').AsFloat;
3693                             Inv[2] := Inv[2] + t_T1_SS7.FieldByName('VS Inv').AsFloat;
3694                           end;
3695                    Inv[3] := Inv[1] - Inv[2];
```

```
3696                for I := 1 to 3 do Inv[I] := Inv[I] / 2.0;
3697                WriteSS7Act(dm_ICM.t_A_SS7H, CLLI, 'PORTLIDBCOND', '244100', Inv);
3698
3699                //RFPC0018 Port at LIDB Reg STP - Poles
3700                if GetSS7Temp(dm_ICM.t_SS7_SSP.FieldByName('Local_STP1').AsString, 'LIDBPORTPOLE')
3701                  then begin
3702                         Inv[1] := t_T1_SS7.FieldByName('Total Inv').AsFloat;
3703                         Inv[2] := t_T1_SS7.FieldByName('VS Inv').AsFloat;
3704                       end
3705                  else begin
3706                         Inv[1] := 0;
3707                         Inv[2] := 0;
3708                       end;
3709                if GetSS7Temp(dm_ICM.t_SS7_SSP.FieldByName('Local_STP2').AsString, 'LIDBPORTPOLE')
3710                  then begin
3711                         Inv[1] := Inv[1] + t_T1_SS7.FieldByName('Total Inv').AsFloat;
3712                         Inv[2] := Inv[2] + t_T1_SS7.FieldByName('VS Inv').AsFloat;
3713                       end;
3714                Inv[3] := Inv[1] - Inv[2];
3715                for I := 1 to 3 do Inv[I] := Inv[I] / 2.0;
3716                WriteSS7Act(dm_ICM.t_A_SS7H, CLLI, 'PORTLIDBPOLE', '241100', Inv);
3717
3718                //RFPC0018 SSP - STP A Link - Lease Expense
3719                if GetSS7Temp(CLLI, 'ALINKEXP')
3720                  then begin
3721                         Inv[1] := t_T1_SS7.FieldByName('Total Inv').AsFloat;
3722                         Inv[2] := t_T1_SS7.FieldByName('VS Inv').AsFloat;
3723                       end
3724                  else begin
3725                         Inv[1] := 0;
3726                         Inv[2] := 0;
3727                       end;
3728                Inv[3] := Inv[1] - Inv[2];
3729                WriteSS7Act(dm_ICM.t_A_SS7H, CLLI, 'ALINKEXP', '621214', Inv);
3730
3731                //RFPC0018 SSP - STP A Link - Aerial Fiber
3732                if GetSS7Temp(CLLI, 'ALINKDISTANCE')
3733                  then Inv[1] := t_T1_SS7.FieldByName('Total Inv').AsFloat
3734                  else Inv[1] := 0;
```

```
3735
3736                Inv[2] := Inv[1];
3737                Inv[3] := Inv[1];
3738                WriteSS7Act(dm_ICM.t_A_SS7H, CLLI, 'ALINKDISTANCE', '000000', Inv);
3739
3740                //RFPC0018 SSP - STP A Link - Circuit equip
3741                if GetSS7Temp(CLLI, 'ALINKCKT')
3742                  then begin
3743                         Inv[1] := t_T1_SS7.FieldByName('Total Inv').AsFloat;
3744                         Inv[2] := t_T1_SS7.FieldByName('VS Inv').AsFloat;
3745                       end
3746                  else begin
3747                         Inv[1] := 0;
3748                         Inv[2] := 0;
3749                       end;
3750                Inv[3] := Inv[1] - Inv[2];
3751                WriteSS7Act(dm_ICM.t_A_SS7H, CLLI, 'ALINKCKT', '223200', Inv);
3752
3753                //RFPC0018 Local - Regional D Link - Lease Expense
3754                if GetSS7Temp(dm_ICM.t_SS7_SSP.FieldByName('Local_STP1').AsString, 'DLINKEXP')
3755                  then begin
3756                         Inv[1] := t_T1_SS7.FieldByName('Total Inv').AsFloat;
3757                         Inv[2] := t_T1_SS7.FieldByName('VS Inv').AsFloat;
3758                       end
3759                  else begin
3760                         Inv[1] := 0;
3761                         Inv[2] := 0;
3762                       end;
3763                if GetSS7Temp(dm_ICM.t_SS7_SSP.FieldByName('Local_STP2').AsString, 'DLINKEXP')
3764                  then begin
3765                         Inv[1] := Inv[1] + t_T1_SS7.FieldByName('Total Inv').AsFloat;
3766                         Inv[2] := Inv[2] + t_T1_SS7.FieldByName('VS Inv').AsFloat;
3767                       end;
3768                Inv[3] := Inv[1] - Inv[2];
3769                for I := 1 to 3 do Inv[I] := Inv[I] / 2.0;
3770                WriteSS7Act(dm_ICM.t_A_SS7H, CLLI, 'DLINKEXP', '242100', Inv);
3771
3772                //RFPC0018 Local - Regional D Link - Aerial Fiber
3773                if GetSS7Temp(dm_ICM.t_SS7_SSP.FieldByName('Local_STP1').AsString, 'DLINKAERF')
```

```
3774                    then begin
3775                          Inv[1] := t_T1_SS7.FieldByName('Total Inv').AsFloat;
3776                          Inv[2] := t_T1_SS7.FieldByName('VS Inv').AsFloat;
3777                    end
3778                    else begin
3779                          Inv[1] := 0;
3780                          Inv[2] := 0;
3781                    end;
3782                  if GetSS7Temp(dm_ICM.t_SS7_SSP.FieldByName('Local_STP2').AsString, 'DLINKAERF')
3783                    then begin
3784                          Inv[1] := Inv[1] + t_T1_SS7.FieldByName('Total Inv').AsFloat;
3785                          Inv[2] := Inv[2] + t_T1_SS7.FieldByName('VS Inv').AsFloat;
3786                    end;
3787                  Inv[3] := Inv[1] - Inv[2];
3788                  for I := 1 to 3 do Inv[I] := Inv[I] / 2.0;
3789                  WriteSS7Act(dm_ICM.t_A_SS7H, CLLI, 'DLINKAERF', '242100', Inv);
3790
3791                  //RFPC0018 Local - Regional D Link - Buried Fiber
3792                  if GetSS7Temp(dm_ICM.t_SS7_SSP.FieldByName('Local_STP1').AsString, 'DLINKBURF')
3793                    then begin
3794                          Inv[1] := t_T1_SS7.FieldByName('Total Inv').AsFloat;
3795                          Inv[2] := t_T1_SS7.FieldByName('VS Inv').AsFloat;
3796                    end
3797                    else begin
3798                          Inv[1] := 0;
3799                          Inv[2] := 0;
3800                    end;
3801                  if GetSS7Temp(dm_ICM.t_SS7_SSP.FieldByName('Local_STP2').AsString, 'DLINKBURF')
3802                    then begin
3803                          Inv[1] := Inv[1] + t_T1_SS7.FieldByName('Total Inv').AsFloat;
3804                          Inv[2] := Inv[2] + t_T1_SS7.FieldByName('VS Inv').AsFloat;
3805                    end;
3806                  Inv[3] := Inv[1] - Inv[2];
3807                  for I := 1 to 3 do Inv[I] := Inv[I] / 2.0;
3808                  WriteSS7Act(dm_ICM.t_A_SS7H, CLLI, 'DLINKBURF', '242300', Inv);
3809
3810                  //RFPC0018 Local - Regional D Link - Underground Fiber
3811                  if GetSS7Temp(dm_ICM.t_SS7_SSP.FieldByName('Local_STP1').AsString, 'DLINKUNDF')
3812                    then begin
```

```
3813                    Inv[1] := t_T1_SS7.FieldByName('Total Inv').AsFloat;
3814                    Inv[2] := t_T1_SS7.FieldByName('VS Inv').AsFloat;
3815                  end
3816                else begin
3817                    Inv[1] := 0;
3818                    Inv[2] := 0;
3819                  end;
3820              if GetSS7Temp(dm_ICM.t_SS7_SSP.FieldByName('Local_STP2').AsString, 'DLINKUNDF')
3821                then begin
3822                    Inv[1] := Inv[1] + t_T1_SS7.FieldByName('Total Inv').AsFloat;
3823                    Inv[2] := Inv[2] + t_T1_SS7.FieldByName('VS Inv').AsFloat;
3824                  end;
3825              Inv[3] := Inv[1] - Inv[2];
3826              for I := 1 to 3 do Inv[I] := Inv[I] / 2.0;
3827              WriteSS7Act(dm_ICM.t_A_SS7H, CLLI, 'DLINKUNDF', '242220', Inv);
3828
3829              //RFPC0018 Local - Regional D Link - Circuit equip
3830              if GetSS7Temp(dm_ICM.t_SS7_SSP.FieldByName('Local_STP1').AsString, 'DLINKCKT')
3831                then begin
3832                    Inv[1] := t_T1_SS7.FieldByName('Total Inv').AsFloat;
3833                    Inv[2] := t_T1_SS7.FieldByName('VS Inv').AsFloat;
3834                  end
3835                else begin
3836                    Inv[1] := 0;
3837                    Inv[2] := 0;
3838                  end;
3839              if GetSS7Temp(dm_ICM.t_SS7_SSP.FieldByName('Local_STP2').AsString, 'DLINKCKT')
3840                then begin
3841                    Inv[1] := Inv[1] + t_T1_SS7.FieldByName('Total Inv').AsFloat;
3842                    Inv[2] := Inv[2] + t_T1_SS7.FieldByName('VS Inv').AsFloat;
3843                  end;
3844              Inv[3] := Inv[1] - Inv[2];
3845              for I := 1 to 3 do Inv[I] := Inv[I] / 2.0;
3846              WriteSS7Act(dm_ICM.t_A_SS7H, CLLI, 'DLINKCKT', '223200', Inv);
3847
3848              //RFPC0018 Local - Regional D Link - Conduit
3849              if GetSS7Temp(dm_ICM.t_SS7_SSP.FieldByName('Local_STP1').AsString, 'DLINKCOND')
3850                then begin
3851                    Inv[1] := t_T1_SS7.FieldByName('Total Inv').AsFloat;
```

```
3852                    Inv[2] := t_T1_SS7.FieldByName('VS Inv').AsFloat;
3853                  end
3854                else begin
3855                    Inv[1] := 0;
3856                    Inv[2] := 0;
3857                  end;
3858              if GetSS7Temp(dm_ICM.t_SS7_SSP.FieldByName('Local_STP2').AsString, 'DLINKCOND')
3859                then begin
3860                    Inv[1] := Inv[1] + t_T1_SS7.FieldByName('Total Inv').AsFloat;
3861                    Inv[2] := Inv[2] + t_T1_SS7.FieldByName('VS Inv').AsFloat;
3862                  end;
3863              Inv[3] := Inv[1] - Inv[2];
3864              for I := 1 to 3 do Inv[I] := Inv[I] / 2.0;
3865              WriteSS7Act(dm_ICM.t_A_SS7H, CLLI, 'DLINKCOND', '244100', Inv);
3866
3867              //RFPC0018 Local - Regional D Link - Poles
3868              if GetSS7Temp(dm_ICM.t_SS7_SSP.FieldByName('Local_STP1').AsString, 'DLINKPOLE')
3869                then begin
3870                    Inv[1] := t_T1_SS7.FieldByName('Total Inv').AsFloat;
3871                    Inv[2] := t_T1_SS7.FieldByName('VS Inv').AsFloat;
3872                  end
3873                else begin
3874                    Inv[1] := 0;
3875                    Inv[2] := 0;
3876                  end;
3877              if GetSS7Temp(dm_ICM.t_SS7_SSP.FieldByName('Local_STP2').AsString, 'DLINKPOLE')
3878                then begin
3879                    Inv[1] := Inv[1] + t_T1_SS7.FieldByName('Total Inv').AsFloat;
3880                    Inv[2] := Inv[2] + t_T1_SS7.FieldByName('VS Inv').AsFloat;
3881                  end;
3882              Inv[3] := Inv[1] - Inv[2];
3883              for I := 1 to 3 do Inv[I] := Inv[I] / 2.0;
3884              WriteSS7Act(dm_ICM.t_A_SS7H, CLLI, 'DLINKPOLE', '241100', Inv);
3885
3886               //RFPC0018 D Link to LIDB Regional STP - Circuit equip
3887              if GetSS7Temp(dm_ICM.t_SS7_SSP.FieldByName('Local_STP1').AsString, 'LIDBLINKCKT')
3888                then begin
3889                    Inv[1] := t_T1_SS7.FieldByName('Total Inv').AsFloat;
3890                    Inv[2] := t_T1_SS7.FieldByName('VS Inv').AsFloat;
```

```
3891                    end
3892                  else begin
3893                    Inv[1] := 0;
3894                    Inv[2] := 0;
3895                    end;
3896                if GetSS7Temp(dm_ICM.t_SS7_SSP.FieldByName('Local_STP2').AsString, 'LIDBLINKCKT')
3897                  then begin
3898                    Inv[1] := Inv[1] + t_T1_SS7.FieldByName('Total Inv').AsFloat;
3899                    Inv[2] := Inv[2] + t_T1_SS7.FieldByName('VS Inv').AsFloat;
3900                    end;
3901                Inv[3] := Inv[1] - Inv[2];
3902                for I := 1 to 3 do Inv[I] := Inv[I] / 2.0;
3903                WriteSS7Act(dm_ICM.t_A_SS7H, CLLI, 'LIDBLINKCKT', '223200', Inv);
3904
3905                //RFPC0018 D Link to LIDB Regional STP - Lease Expense
3906                if GetSS7Temp(dm_ICM.t_SS7_SSP.FieldByName('Local_STP1').AsString, 'LIDBLINKEXP')
3907                  then begin
3908                    Inv[1] := t_T1_SS7.FieldByName('Total Inv').AsFloat;
3909                    Inv[2] := t_T1_SS7.FieldByName('VS Inv').AsFloat;
3910                    end
3911                  else begin
3912                    Inv[1] := 0;
3913                    Inv[2] := 0;
3914                    end;
3915                if GetSS7Temp(dm_ICM.t_SS7_SSP.FieldByName('Local_STP2').AsString, 'LIDBLINKEXP')
3916                  then begin
3917                    Inv[1] := Inv[1] + t_T1_SS7.FieldByName('Total Inv').AsFloat;
3918                    Inv[2] := Inv[2] + t_T1_SS7.FieldByName('VS Inv').AsFloat;
3919                    end;
3920                Inv[3] := Inv[1] - Inv[2];
3921                for I := 1 to 3 do Inv[I] := Inv[I] / 2.0;
3922                WriteSS7Act(dm_ICM.t_A_SS7H, CLLI, 'LIDBLINKEXP', '621214', Inv);
3923
3924                //RFPC0018 Regional STP Port - Aerial Fiber
3925                if GetSS7Temp(dm_ICM.t_SS7_SSP.FieldByName('Local_STP1').AsString, 'RSPORTAERF')
3926                  then begin
3927                    Inv[1] := t_T1_SS7.FieldByName('Total Inv').AsFloat;
3928                    Inv[2] := t_T1_SS7.FieldByName('VS Inv').AsFloat;
3929                    end
```

```
3930                  else begin
3931                        Inv[1] := 0;
3932                        Inv[2] := 0;
3933                      end;
3934                if GetSS7Temp(dm_ICM.t_SS7_SSP.FieldByName('Local_STP2').AsString, 'RSPORTAERF')
3935                  then begin
3936                        Inv[1] := Inv[1] + t_T1_SS7.FieldByName('Total Inv').AsFloat;
3937                        Inv[2] := Inv[2] + t_T1_SS7.FieldByName('VS Inv').AsFloat;
3938                      end;
3939                Inv[3] := Inv[1] - Inv[2];
3940                for I := 1 to 3 do Inv[I] := Inv[I] / 2.0;
3941                WriteSS7Act(dm_ICM.t_A_SS7H, CLLI, 'RSPORTAERF', '242100', Inv);
3942
3943                //RFPC0018 Regional STP Port - Buried Fiber
3944                if GetSS7Temp(dm_ICM.t_SS7_SSP.FieldByName('Local_STP1').AsString, 'RSPORTBURF')
3945                  then begin
3946                        Inv[1] := t_T1_SS7.FieldByName('Total Inv').AsFloat;
3947                        Inv[2] := t_T1_SS7.FieldByName('VS Inv').AsFloat;
3948                      end
3949                  else begin
3950                        Inv[1] := 0;
3951                        Inv[2] := 0;
3952                      end;
3953                if GetSS7Temp(dm_ICM.t_SS7_SSP.FieldByName('Local_STP2').AsString, 'RSPORTBURF')
3954                  then begin
3955                        Inv[1] := Inv[1] + t_T1_SS7.FieldByName('Total Inv').AsFloat;
3956                        Inv[2] := Inv[2] + t_T1_SS7.FieldByName('VS Inv').AsFloat;
3957                      end;
3958                Inv[3] := Inv[1] - Inv[2];
3959                for I := 1 to 3 do Inv[I] := Inv[I] / 2.0;
3960                WriteSS7Act(dm_ICM.t_A_SS7H, CLLI, 'RSPORTBURF', '242300', Inv);
3961
3962                //RFPC0018 Regional STP Port - Underground Fiber
3963                if GetSS7Temp(dm_ICM.t_SS7_SSP.FieldByName('Local_STP1').AsString, 'RSPORTUNDF')
3964                  then begin
3965                        Inv[1] := t_T1_SS7.FieldByName('Total Inv').AsFloat;
3966                        Inv[2] := t_T1_SS7.FieldByName('VS Inv').AsFloat;
3967                      end
3968                  else begin
```

```
3969                    Inv[1] := 0;
3970                    Inv[2] := 0;
3971                  end;
3972          if GetSS7Temp(dm_ICM.t_SS7_SSP.FieldByName('Local_STP2').AsString, 'RSPORTUNDF')
3973            then begin
3974                    Inv[1] := Inv[1] + t_T1_SS7.FieldByName('Total Inv').AsFloat;
3975                    Inv[2] := Inv[2] + t_T1_SS7.FieldByName('VS Inv').AsFloat;
3976                  end;
3977          Inv[3] := Inv[1] - Inv[2];
3978          for I := 1 to 3 do Inv[I] := Inv[I] / 2.0;
3979          WriteSS7Act(dm_ICM.t_A_SS7H, CLLI, 'RSPORTUNDF', '242220', Inv);
3980
3981          //RFPC0018 Regional STP Port - Circuit equip
3982          if GetSS7Temp(dm_ICM.t_SS7_SSP.FieldByName('Local_STP1').AsString, 'RSPORTCKT')
3983            then begin
3984                    Inv[1] := t_T1_SS7.FieldByName('Total Inv').AsFloat;
3985                    Inv[2] := t_T1_SS7.FieldByName('VS Inv').AsFloat;
3986                  end
3987            else begin
3988                    Inv[1] := 0;
3989                    Inv[2] := 0;
3990                  end;
3991          if GetSS7Temp(dm_ICM.t_SS7_SSP.FieldByName('Local_STP2').AsString, 'RSPORTCKT')
3992            then begin
3993                    Inv[1] := Inv[1] + t_T1_SS7.FieldByName('Total Inv').AsFloat;
3994                    Inv[2] := Inv[2] + t_T1_SS7.FieldByName('VS Inv').AsFloat;
3995                  end;
3996          Inv[3] := Inv[1] - Inv[2];
3997          for I := 1 to 3 do Inv[I] := Inv[I] / 2.0;
3998          WriteSS7Act(dm_ICM.t_A_SS7H, CLLI, 'RSPORTCKT', '223200', Inv);
3999
4000          //RFPC0018 Regional STP Port - Switching Equip
4001          if GetSS7Temp(dm_ICM.t_SS7_SSP.FieldByName('Local_STP1').AsString, 'RSPORTSWT')
4002            then begin
4003                    Inv[1] := t_T1_SS7.FieldByName('Total Inv').AsFloat;
4004                    Inv[2] := t_T1_SS7.FieldByName('VS Inv').AsFloat;
4005                  end
4006            else begin
4007                    Inv[1] := 0;
```

```
4008                    Inv[2] := 0;
4009                  end;
4010          if GetSS7Temp(dm_ICM.t_SS7_SSP.FieldByName('Local_STP2').AsString, 'RSPORTSWT')
4011            then begin
4012                    Inv[1] := Inv[1] + t_T1_SS7.FieldByName('Total Inv').AsFloat;
4013                    Inv[2] := Inv[2] + t_T1_SS7.FieldByName('VS Inv').AsFloat;
4014                  end;
4015          Inv[3] := Inv[1] - Inv[2];
4016          for I := 1 to 3 do Inv[I] := Inv[I] / 2.0;
4017          WriteSS7Act(dm_ICM.t_A_SS7H, CLLI, 'RSPORTSWT', '221200', Inv);
4018
4019          //RFPC0018 Regional STP Port - Conduit
4020          if GetSS7Temp(dm_ICM.t_SS7_SSP.FieldByName('Local_STP1').AsString, 'RSPORTCOND')
4021            then begin
4022                    Inv[1] := t_T1_SS7.FieldByName('Total Inv').AsFloat;
4023                    Inv[2] := t_T1_SS7.FieldByName('VS Inv').AsFloat;
4024                  end
4025            else begin
4026                    Inv[1] := 0;
4027                    Inv[2] := 0;
4028                  end;
4029          if GetSS7Temp(dm_ICM.t_SS7_SSP.FieldByName('Local_STP2').AsString, 'RSPORTCOND')
4030            then begin
4031                    Inv[1] := Inv[1] + t_T1_SS7.FieldByName('Total Inv').AsFloat;
4032                    Inv[2] := Inv[2] + t_T1_SS7.FieldByName('VS Inv').AsFloat;
4033                  end;
4034          Inv[3] := Inv[1] - Inv[2];
4035          for I := 1 to 3 do Inv[I] := Inv[I] / 2.0;
4036          WriteSS7Act(dm_ICM.t_A_SS7H, CLLI, 'RSPORTCOND', '244100', Inv);
4037
4038          //RFPC0018 Regional STP Port - Poles
4039          if GetSS7Temp(dm_ICM.t_SS7_SSP.FieldByName('Local_STP1').AsString, 'RSPORTPOLE')
4040            then begin
4041                    Inv[1] := t_T1_SS7.FieldByName('Total Inv').AsFloat;
4042                    Inv[2] := t_T1_SS7.FieldByName('VS Inv').AsFloat;
4043                  end
4044            else begin
4045                    Inv[1] := 0;
4046                    Inv[2] := 0;
```

```
4047                    end;
4048                if GetSS7Temp(dm_ICM.t_SS7_SSP.FieldByName('Local_STP2').AsString, 'RSPORTPOLE')
4049                  then begin
4050                         Inv[1] := Inv[1] + t_T1_SS7.FieldByName('Total Inv').AsFloat;
4051                         Inv[2] := Inv[2] + t_T1_SS7.FieldByName('VS Inv').AsFloat;
4052                    end;
4053                Inv[3] := Inv[1] - Inv[2];
4054                for I := 1 to 3 do Inv[I] := Inv[I] / 2.0;
4055                WriteSS7Act(dm_ICM.t_A_SS7H, CLLI, 'RSPORTPOLE', '241100', Inv);
4056
4057                //RFPC0018 Local  STP Port - Aerial Fiber
4058                if GetSS7Temp(dm_ICM.t_SS7_SSP.FieldByName('Local_STP1').AsString, 'STPPORTAERF')
4059                  then begin
4060                         Inv[1] := t_T1_SS7.FieldByName('Total Inv').AsFloat;
4061                         Inv[2] := t_T1_SS7.FieldByName('VS Inv').AsFloat;
4062                    end
4063                  else begin
4064                         Inv[1] := 0;
4065                         Inv[2] := 0;
4066                    end;
4067                if GetSS7Temp(dm_ICM.t_SS7_SSP.FieldByName('Local_STP2').AsString, 'STPPORTAERF')
4068                  then begin
4069                         Inv[1] := Inv[1] + t_T1_SS7.FieldByName('Total Inv').AsFloat;
4070                         Inv[2] := Inv[2] + t_T1_SS7.FieldByName('VS Inv').AsFloat;
4071                    end;
4072                Inv[3] := Inv[1] - Inv[2];
4073                WriteSS7Act(dm_ICM.t_A_SS7H, CLLI, 'STPPORTAERF', '242100', Inv);
4074
4075                //RFPC0018 Local  STP Port - Buried Fiber
4076                if GetSS7Temp(dm_ICM.t_SS7_SSP.FieldByName('Local_STP1').AsString, 'STPPORTBURF')
4077                  then begin
4078                         Inv[1] := t_T1_SS7.FieldByName('Total Inv').AsFloat;
4079                         Inv[2] := t_T1_SS7.FieldByName('VS Inv').AsFloat;
4080                    end
4081                  else begin
4082                         Inv[1] := 0;
4083                         Inv[2] := 0;
4084                    end;
4085                if GetSS7Temp(dm_ICM.t_SS7_SSP.FieldByName('Local_STP2').AsString, 'STPPORTBURF')
```

```
4086                    then begin
4087                          Inv[1] := Inv[1] + t_T1_SS7.FieldByName('Total Inv').AsFloat;
4088                          Inv[2] := Inv[2] + t_T1_SS7.FieldByName('VS Inv').AsFloat;
4089                        end;
4090                  Inv[3] := Inv[1] - Inv[2];
4091                  WriteSS7Act(dm_ICM.t_A_SS7H, CLLI, 'STPPORTBURF', '242300', Inv);
4092
4093                  //RFPC0018 Local STP Port - Underground Fiber
4094                  if GetSS7Temp(dm_ICM.t_SS7_SSP.FieldByName('Local_STP1').AsString, 'STPPORTUNDF')
4095                    then begin
4096                          Inv[1] := t_T1_SS7.FieldByName('Total Inv').AsFloat;
4097                          Inv[2] := t_T1_SS7.FieldByName('VS Inv').AsFloat;
4098                        end
4099                    else begin
4100                          Inv[1] := 0;
4101                          Inv[2] := 0;
4102                        end;
4103                  if GetSS7Temp(dm_ICM.t_SS7_SSP.FieldByName('Local_STP2').AsString, 'STPPORTUNDF')
4104                    then begin
4105                          Inv[1] := Inv[1] + t_T1_SS7.FieldByName('Total Inv').AsFloat;
4106                          Inv[2] := Inv[2] + t_T1_SS7.FieldByName('VS Inv').AsFloat;
4107                        end;
4108                  Inv[3] := Inv[1] - Inv[2];
4109                  WriteSS7Act(dm_ICM.t_A_SS7H, CLLI, 'STPPORTUNDF', '242220', Inv);
4110
4111                  //RFPC0018 Local STP Port - Circuit equip
4112                  if GetSS7Temp(dm_ICM.t_SS7_SSP.FieldByName('Local_STP1').AsString, 'STPPORTCKT')
4113                    then begin
4114                          Inv[1] := t_T1_SS7.FieldByName('Total Inv').AsFloat;
4115                          Inv[2] := t_T1_SS7.FieldByName('VS Inv').AsFloat;
4116                        end
4117                    else begin
4118                          Inv[1] := 0;
4119                          Inv[2] := 0;
4120                        end;
4121                  if GetSS7Temp(dm_ICM.t_SS7_SSP.FieldByName('Local_STP2').AsString, 'STPPORTCKT')
4122                    then begin
4123                          Inv[1] := Inv[1] + t_T1_SS7.FieldByName('Total Inv').AsFloat;
4124                          Inv[2] := Inv[2] + t_T1_SS7.FieldByName('VS Inv').AsFloat;
```

```
4125                    end;
4126                Inv[3] := Inv[1] - Inv[2];
4127                WriteSS7Act(dm_ICM.t_A_SS7H, CLLI, 'STPPORTCKT', '223200', Inv);
4128
4129                //RFPC0018 Local STP Port - Switching Equip
4130                if GetSS7Temp(dm_ICM.t_SS7_SSP.FieldByName('Local_STP1').AsString, 'STPPORTSWT')
4131                  then begin
4132                         Inv[1] := t_T1_SS7.FieldByName('Total Inv').AsFloat;
4133                         Inv[2] := t_T1_SS7.FieldByName('VS Inv').AsFloat;
4134                       end
4135                  else begin
4136                         Inv[1] := 0;
4137                         Inv[2] := 0;
4138                       end;
4139                if GetSS7Temp(dm_ICM.t_SS7_SSP.FieldByName('Local_STP2').AsString, 'STPPORTSWT')
4140                  then begin
4141                         Inv[1] := Inv[1] + t_T1_SS7.FieldByName('Total Inv').AsFloat;
4142                         Inv[2] := Inv[2] + t_T1_SS7.FieldByName('VS Inv').AsFloat;
4143                       end;
4144                Inv[3] := Inv[1] - Inv[2];
4145                WriteSS7Act(dm_ICM.t_A_SS7H, CLLI, 'STPPORTSWT', '221200', Inv);
4146
4147                //RFPC0018 Local STP Port - Conduit
4148                if GetSS7Temp(dm_ICM.t_SS7_SSP.FieldByName('Local_STP1').AsString, 'STPPORTCOND')
4149                  then begin
4150                         Inv[1] := t_T1_SS7.FieldByName('Total Inv').AsFloat;
4151                         Inv[2] := t_T1_SS7.FieldByName('VS Inv').AsFloat;
4152                       end
4153                  else begin
4154                         Inv[1] := 0;
4155                         Inv[2] := 0;
4156                       end;
4157                if GetSS7Temp(dm_ICM.t_SS7_SSP.FieldByName('Local_STP2').AsString, 'STPPORTCOND')
4158                  then begin
4159                         Inv[1] := Inv[1] + t_T1_SS7.FieldByName('Total Inv').AsFloat;
4160                         Inv[2] := Inv[2] + t_T1_SS7.FieldByName('VS Inv').AsFloat;
4161                       end;
4162                Inv[3] := Inv[1] - Inv[2];
4163                WriteSS7Act(dm_ICM.t_A_SS7H, CLLI, 'STPPORTCOND', '244100', Inv);
```

```
4164
4165                    //RFPC0018 Local STP Port - Poles
4166                    if GetSS7Temp(dm_ICM.t_SS7_SSP.FieldByName('Local_STP1').AsString, 'STPPORTPOLE')
4167                      then begin
4168                             Inv[1] := t_T1_SS7.FieldByName('Total Inv').AsFloat;
4169                             Inv[2] := t_T1_SS7.FieldByName('VS Inv').AsFloat;
4170                           end
4171                      else begin
4172                             Inv[1] := 0;
4173                             Inv[2] := 0;
4174                           end;
4175                    if GetSS7Temp(dm_ICM.t_SS7_SSP.FieldByName('Local_STP2').AsString, 'STPPORTPOLE')
4176                      then begin
4177                             Inv[1] := Inv[1] + t_T1_SS7.FieldByName('Total Inv').AsFloat;
4178                             Inv[2] := Inv[2] + t_T1_SS7.FieldByName('VS Inv').AsFloat;
4179                           end;
4180                    Inv[3] := Inv[1] - Inv[2];
4181                    WriteSS7Act(dm_ICM.t_A_SS7H, CLLI, 'STPPORTPOLE', '241100', Inv);
4182
4183                    //RFPC0018 Software per TCAP Message at Local STP
4184                    if GetSS7Temp(dm_ICM.t_SS7_SSP.FieldByName('Local_STP1').AsString, 'TCAPSWL')
4185                      then begin
4186                             Inv[1] := t_T1_SS7.FieldByName('Total Inv').AsFloat;
4187                             Inv[2] := t_T1_SS7.FieldByName('VS Inv').AsFloat;
4188                           end
4189                      else begin
4190                             Inv[1] := 0;
4191                             Inv[2] := 0;
4192                           end;
4193                    if GetSS7Temp(dm_ICM.t_SS7_SSP.FieldByName('Local_STP2').AsString, 'TCAPSWL')
4194                      then begin
4195                             Inv[1] := Inv[1] + t_T1_SS7.FieldByName('Total Inv').AsFloat;
4196                             Inv[2] := Inv[2] + t_T1_SS7.FieldByName('VS Inv').AsFloat;
4197                           end;
4198                    Inv[3] := Inv[1] - Inv[2];
4199                    for I := 1 to 3 do Inv[I] := Inv[I] / 2.0;
4200                    WriteSS7Act(dm_ICM.t_A_SS7H, CLLI, 'TCAPSWL', '221200', Inv);
4201
4202                    //RFPC0018 SCP investment per 800 Database Query
```

```
4203                    if GetSS7Temp('SCP0XX00000', 'SCP800')
4204                      then begin
4205                             Inv[1] := t_T1_SS7.FieldByName('Total Inv').AsFloat;
4206                             Inv[2] := t_T1_SS7.FieldByName('VS Inv').AsFloat;
4207                           end
4208                      else begin
4209                             Inv[1] := 0;
4210                             Inv[2] := 0;
4211                           end;
4212                    Inv[3] := Inv[1] - Inv[2];
4213                    WriteSS7Act(dm_ICM.t_A_SS7H, CLLI, 'SCP800', '221200', Inv);
4214
4215                    //RFPC0018 SCP investment per LIDB Query
4216                    if GetSS7Temp('SCP0XX00000', 'SCPLIDB')
4217                      then begin
4218                             Inv[1] := t_T1_SS7.FieldByName('Total Inv').AsFloat;
4219                             Inv[2] := t_T1_SS7.FieldByName('VS Inv').AsFloat;
4220                           end
4221                      else begin
4222                             Inv[1] := 0;
4223                             Inv[2] := 0;
4224                           end;
4225                    Inv[3] := Inv[1] - Inv[2];
4226                    WriteSS7Act(dm_ICM.t_A_SS7H, CLLI, 'SCPLIDB', '221200', Inv);
4227
4228                    //RFPC0018 SSP investment per 800 database SW Port
4229                    if GetSS7Temp(CLLI, 'SSP800PORT')
4230                      then begin
4231                             Inv[1] := t_T1_SS7.FieldByName('Total Inv').AsFloat;
4232                             Inv[2] := t_T1_SS7.FieldByName('VS Inv').AsFloat;
4233                           end
4234                      else begin
4235                             Inv[1] := 0;
4236                             Inv[2] := 0;
4237                           end;
4238                    Inv[3] := Inv[1] - Inv[2];
4239                    WriteSS7Act(dm_ICM.t_A_SS7H, CLLI, 'SSP800PORT', '221200', Inv);
4240
4241                    //RFPC0018 SSP investment per LIDB SW Port
```

```
4242                if GetSS7Temp(CLLI, 'SSPLIDBPORT')
4243                  then begin
4244                         Inv[1] := t_T1_SS7.FieldByName('Total Inv').AsFloat;
4245                         Inv[2] := t_T1_SS7.FieldByName('VS Inv').AsFloat;
4246                       end
4247                  else begin
4248                         Inv[1] := 0;
4249                         Inv[2] := 0;
4250                       end;
4251                Inv[3] := Inv[1] - Inv[2];
4252                WriteSS7Act(dm_ICM.t_A_SS7H, CLLI, 'SSPLIDBPORT', '221200', Inv);
4253
4254                //RFPC0018 SSP investment per Port
4255                if GetSS7Temp(CLLI, 'SSPPORT')
4256                  then begin
4257                         Inv[1] := t_T1_SS7.FieldByName('Total Inv').AsFloat;
4258                         Inv[2] := t_T1_SS7.FieldByName('VS Inv').AsFloat;
4259                       end
4260                  else begin
4261                         Inv[1] := 0;
4262                         Inv[2] := 0;
4263                       end;
4264                Inv[3] := Inv[1] - Inv[2];
4265                WriteSS7Act(dm_ICM.t_A_SS7H, CLLI, 'SSPPORT', '221200', Inv);
4266
4267                //RFPC0018 Find Local STP1 CLLI code in STP table
4268                if dm_ICM.t_SS7_STP.FindKey([dm_ICM.t_SS7_SSP.FieldByName('Local_STP1').AsString])
4269                  then begin
4270
4271                         //RFPC0018 Software per TCAP Message at Regional STP
4272                         if GetSS7Temp(dm_ICM.t_SS7_STP.FieldByName('Regional_STP1').AsString, 'TCAPSWR')
4273                           then begin
4274                                  Inv[1] := t_T1_SS7.FieldByName('Total Inv').AsFloat;
4275                                  Inv[2] := t_T1_SS7.FieldByName('VS Inv').AsFloat;
4276                                end
4277                           else begin
4278                                  Inv[1] := 0;
4279                                  Inv[2] := 0;
4280                                end;
```

```
4281                          Inv[3] := Inv[1] - Inv[2];
4282                          WriteSS7Act(dm_ICM.t_A_SS7H, CLLI, 'TCAPSWR', '241100', Inv);
4283
4284                          //RFPC0018 A Link STP to 800 SCP - Aerial Fiber
4285                          if GetSS7Temp(dm_ICM.t_SS7_STP.FieldByName('Regional_STP1').AsString,
4286   'ALINK800AERF')
4287                            then begin
4288                                   Inv[1] := t_T1_SS7.FieldByName('Total Inv').AsFloat;
4289                                   Inv[2] := t_T1_SS7.FieldByName('VS Inv').AsFloat;
4290                                 end
4291                            else begin
4292                                   Inv[1] := 0;
4293                                   Inv[2] := 0;
4294                                 end;
4295                          Inv[3] := Inv[1] - Inv[2];
4296                          WriteSS7Act(dm_ICM.t_A_SS7H, CLLI, 'ALINK800AERF', '242100', Inv);
4297
4298                          //RFPC0018 A Link STP to 800 SCP - Buried Fiber
4299                          if GetSS7Temp(dm_ICM.t_SS7_STP.FieldByName('Regional_STP1').AsString,
4300   'ALINK800BURF')
4301                            then begin
4302                                   Inv[1] := t_T1_SS7.FieldByName('Total Inv').AsFloat;
4303                                   Inv[2] := t_T1_SS7.FieldByName('VS Inv').AsFloat;
4304                                 end
4305                            else begin
4306                                   Inv[1] := 0;
4307                                   Inv[2] := 0;
4308                                 end;
4309                          Inv[3] := Inv[1] - Inv[2];
4310                          WriteSS7Act(dm_ICM.t_A_SS7H, CLLI, 'ALINK800BURF', '242300', Inv);
4311
4312                          //RFPC0018 A Link STP to 800 SCP - Underground Fiber
4313                          if GetSS7Temp(dm_ICM.t_SS7_STP.FieldByName('Regional_STP1').AsString,
4314   'ALINK800UNDF')
4315                            then begin
4316                                   Inv[1] := t_T1_SS7.FieldByName('Total Inv').AsFloat;
4317                                   Inv[2] := t_T1_SS7.FieldByName('VS Inv').AsFloat;
4318                                 end
4319                            else begin
```

```
4320                                Inv[1] := 0;
4321                                Inv[2] := 0;
4322                              end;
4323                          Inv[3] := Inv[1] - Inv[2];
4324                          WriteSS7Act(dm_ICM.t_A_SS7H, CLLI, 'ALINK800UNDF', '242220', Inv);
4325
4326                          //RFPC0018 A Link STP to 800 SCP - Circuit Equip
4327                          if GetSS7Temp(dm_ICM.t_SS7_STP.FieldByName('Regional_STP1').AsString,
4328  'ALINK800CKT')
4329                            then begin
4330                                Inv[1] := t_T1_SS7.FieldByName('Total Inv').AsFloat;
4331                                Inv[2] := t_T1_SS7.FieldByName('VS Inv').AsFloat;
4332                              end
4333                            else begin
4334                                Inv[1] := 0;
4335                                Inv[2] := 0;
4336                              end;
4337                          Inv[3] := Inv[1] - Inv[2];
4338                          WriteSS7Act(dm_ICM.t_A_SS7H, CLLI, 'ALINK800CKT', '223200', Inv);
4339
4340                          //RFPC0018 A Link STP to 800 SCP - COND
4341                          if GetSS7Temp(dm_ICM.t_SS7_STP.FieldByName('Regional_STP1').AsString,
4342  'ALINK800COND')
4343                            then begin
4344                                Inv[1] := t_T1_SS7.FieldByName('Total Inv').AsFloat;
4345                                Inv[2] := t_T1_SS7.FieldByName('VS Inv').AsFloat;
4346                              end
4347                            else begin
4348                                Inv[1] := 0;
4349                                Inv[2] := 0;
4350                              end;
4351                          Inv[3] := Inv[1] - Inv[2];
4352                          WriteSS7Act(dm_ICM.t_A_SS7H, CLLI, 'ALINK800COND', '244100', Inv);
4353
4354                          //RFPC0018 A Link STP to 800 SCP - Poles
4355                          if GetSS7Temp(dm_ICM.t_SS7_STP.FieldByName('Regional_STP1').AsString,
4356  'ALINK800POLE')
4357                            then begin
4358                                Inv[1] := t_T1_SS7.FieldByName('Total Inv').AsFloat;
```

```
4359                                Inv[2] := t_T1_SS7.FieldByName('VS Inv').AsFloat;
4360                              end
4361                            else begin
4362                                Inv[1] := 0;
4363                                Inv[2] := 0;
4364                              end;
4365                          Inv[3] := Inv[1] - Inv[2];
4366                          WriteSS7Act(dm_ICM.t_A_SS7H, CLLI, 'ALINK800POLE', '241100', Inv);
4367
4368                          //RFPC0018 A Link STP to LIDB SCP - Aerial Fiber
4369                          if GetSS7Temp(dm_ICM.t_SS7_STP.FieldByName('Regional_STP1').AsString,
4370  'ALINKLIDBAERF')
4371                            then begin
4372                                Inv[1] := t_T1_SS7.FieldByName('Total Inv').AsFloat;
4373                                Inv[2] := t_T1_SS7.FieldByName('VS Inv').AsFloat;
4374                              end
4375                            else begin
4376                                Inv[1] := 0;
4377                                Inv[2] := 0;
4378                              end;
4379                          Inv[3] := Inv[1] - Inv[2];
4380                          WriteSS7Act(dm_ICM.t_A_SS7H, CLLI, 'ALINKLIDBAERF', '242100', Inv);
4381
4382                          //RFPC0018 A Link STP to LIDB SCP - Buried Fiber
4383                          if GetSS7Temp(dm_ICM.t_SS7_STP.FieldByName('Regional_STP1').AsString,
4384  'ALINKLIDBBURF')
4385                            then begin
4386                                Inv[1] := t_T1_SS7.FieldByName('Total Inv').AsFloat;
4387                                Inv[2] := t_T1_SS7.FieldByName('VS Inv').AsFloat;
4388                              end
4389                            else begin
4390                                Inv[1] := 0;
4391                                Inv[2] := 0;
4392                              end;
4393                          Inv[3] := Inv[1] - Inv[2];
4394                          WriteSS7Act(dm_ICM.t_A_SS7H, CLLI, 'ALINKLIDBBURF', '242300', Inv);
4395
4396                          //RFPC0018 A Link STP to LIDB SCP - Underground Fiber
```

```
4397                        if GetSS7Temp(dm_ICM.t_SS7_STP.FieldByName('Regional_STP1').AsString,
4398  'ALINKLIDBUNDF')
4399                          then begin
4400                                 Inv[1] := t_T1_SS7.FieldByName('Total Inv').AsFloat;
4401                                 Inv[2] := t_T1_SS7.FieldByName('VS Inv').AsFloat;
4402                               end
4403                          else begin
4404                                 Inv[1] := 0;
4405                                 Inv[2] := 0;
4406                               end;
4407                        Inv[3] := Inv[1] - Inv[2];
4408                        WriteSS7Act(dm_ICM.t_A_SS7H, CLLI, 'ALINKLIDBUNDF', '242220', Inv);
4409
4410                        //RFPC0018 A Link STP to LIDB SCP - Circuit Equip
4411                        if GetSS7Temp(dm_ICM.t_SS7_STP.FieldByName('Regional_STP1').AsString,
4412  'ALINKLIDBCKT')
4413                          then begin
4414                                 Inv[1] := t_T1_SS7.FieldByName('Total Inv').AsFloat;
4415                                 Inv[2] := t_T1_SS7.FieldByName('VS Inv').AsFloat;
4416                               end
4417                          else begin
4418                                 Inv[1] := 0;
4419                                 Inv[2] := 0;
4420                               end;
4421                        Inv[3] := Inv[1] - Inv[2];
4422                        WriteSS7Act(dm_ICM.t_A_SS7H, CLLI, 'ALINKLIDBCKT', '223200', Inv);
4423
4424                        //RFPC0018 A Link STP to LIDB SCP - COND
4425                        if GetSS7Temp(dm_ICM.t_SS7_STP.FieldByName('Regional_STP1').AsString,
4426  'ALINKLIDBCOND')
4427                          then begin
4428                                 Inv[1] := t_T1_SS7.FieldByName('Total Inv').AsFloat;
4429                                 Inv[2] := t_T1_SS7.FieldByName('VS Inv').AsFloat;
4430                               end
4431                          else begin
4432                                 Inv[1] := 0;
4433                                 Inv[2] := 0;
4434                               end;
4435                        Inv[3] := Inv[1] - Inv[2];
```

```
4436                          WriteSS7Act(dm_ICM.t_A_SS7H, CLLI, 'ALINKLIDBCOND', '244100', Inv);
4437
4438                          //RFPC0018 A Link STP to LIDB SCP - Poles
4439                          if GetSS7Temp(dm_ICM.t_SS7_STP.FieldByName('Regional_STP1').AsString,
4440   'ALINKLIDBPOLE')
4441                            then begin
4442                                   Inv[1] := t_T1_SS7.FieldByName('Total Inv').AsFloat;
4443                                   Inv[2] := t_T1_SS7.FieldByName('VS Inv').AsFloat;
4444                                 end
4445                            else begin
4446                                   Inv[1] := 0;
4447                                   Inv[2] := 0;
4448                                 end;
4449                          Inv[3] := Inv[1] - Inv[2];
4450                          WriteSS7Act(dm_ICM.t_A_SS7H, CLLI, 'ALINKLIDBPOLE', '241100', Inv);
4451                        end
4452                    else begin
4453                          MessageDlg('Could not find CLLI -'+ CLLI + '- in STP table.', mtError, [mbok], 0);
4454                          WeAreWatching.LogEvent(1,'Process SS7','Could not find CLLI -'+ CLLI + '- in STP
4455   table.');
4456                         end;
4457                end
4458           else begin  //RFPC0018 CLLI code not found
4459                  MessageDlg('Could not find CLLI -'+ CLLI + '- in node table.', mtError, [mbok], 0);
4460                  WeAreWatching.LogEvent(1,'Process SS7','Could not find CLLI -'+ CLLI + '- in node table.');
4461                end;
4462
4463         dm_ICM.t_SS7_SSP.Next;  //RFPC0018 move to next record in the SSP table
4464       end;
4465    end;
4466
4467   procedure Tf_ss7.OutputRemoteInvestments;
4468    Var
4469     SW_Type : String;
4470     CLLI    : String;
4471     HCLLI   : String;
4472     Item    : String;
4473     Account : String;
4474     Inv     : TInvestment;
```

```
4475
4476  begin
4477    dm_ICM.t_Nodes.First;
4478    while Not dm_ICM.t_Nodes.EOF do
4479      begin
4480        // get switch type from nodes table
4481        //RFPC0060 Substitution no allowed. Removed check.
4482        SW_Type := dm_ICM.t_Nodes.FieldByName('Switch Type').AsString;
4483
4484        SW_Type := UpperCase(SW_Type);
4485        if Pos('REM', SW_Type) > 0
4486          then begin
4487                 //RFPC0018 Duplicate hosts investment for remotes
4488                 CLLI  := dm_ICM.t_Nodes.FieldByName('CLLI').AsString;
4489                 HCLLI := dm_ICM.t_Nodes.FieldByName('Host CLLI').AsString;
4490                 dm_ICM.t_A_SS7H.SetRange([HCLLI],[HCLLI]);
4491
4492                 dm_ICM.t_A_SS7H.First;
4493                 while Not dm_ICM.t_A_SS7H.EOF do
4494                   begin
4495                     Account := dm_ICM.t_A_SS7H.FieldByName('Account').AsString;
4496                     Item    := dm_ICM.t_A_SS7H.FieldByName('Component').AsString;
4497                     Inv[1]  := dm_ICM.t_A_SS7H.FieldByName('Total Inv').AsFloat;
4498                     Inv[2]  := dm_ICM.t_A_SS7H.FieldByName('VS Inv').AsFloat;
4499                     Inv[3]  := dm_ICM.t_A_SS7H.FieldByName('VIS Inv').AsFloat;
4500                     WriteSS7Act(dm_ICM.t_A_SS7R, CLLI, Item, Account, Inv);
4501                     dm_ICM.t_A_SS7H.Next;
4502                   end;
4503               end;
4504        dm_ICM.t_Nodes.Next;
4505      end;
4506  end;
4507
4508  function Tf_ss7.GetDistance(CLLI1, CLLI2 : String) : Double;
4509  Var
4510    CurrentRec : TBookmark;
4511    Lat1, Lat2 : Double;
4512    Lng1, Lng2 : Double;
4513
```

```
4514  begin
4515    //RFPC0018 find the airline distance between two CLLI codes
4516    CurrentRec := dm_ICM.t_SS7_STP.GetBookMark;
4517    if dm_ICM.t_SS7_STP.FindKey([CLLI1])   //RFPC0018 Find lat and long for CLLI 1
4518      then begin
4519             Lat1 := dm_ICM.t_SS7_STP.FieldByName('STP_Latitude').AsFloat;
4520             Lng1 := dm_ICM.t_SS7_STP.FieldByName('STP_Longitude').AsFloat;
4521           end
4522      else begin
4523             MessageDlg('Could not find Lat/Long for '+ CLLI1 + '.', mtError, [mbok], 0);
4524             WeAreWatching.LogEvent(1,'Process SS7','Could not find Lat/Long for '+ CLLI1 + '.');
4525             Result := 0.0;
4526             dm_ICM.t_SS7_STP.GotoBookMark(CurrentRec);
4527             dm_ICM.t_SS7_STP.FreeBookMark(CurrentRec);
4528             Exit;
4529           end;
4530
4531    if dm_ICM.t_SS7_STP.FindKey([CLLI2])   //RFPC0018 Find lat and long for CLLI 2
4532      then begin
4533             Lat2 := dm_ICM.t_SS7_STP.FieldByName('STP_Latitude').AsFloat;
4534             Lng2 := dm_ICM.t_SS7_STP.FieldByName('STP_Longitude').AsFloat;
4535           end
4536      else begin
4537             MessageDlg('Could not find Lat/Long for '+ CLLI2 + '.', mtError, [mbok], 0);
4538             WeAreWatching.LogEvent(1,'Process SS7','Could not find Lat/Long for '+ CLLI2 + '.');
4539             Result := 0.0;
4540             dm_ICM.t_SS7_STP.GotoBookMark(CurrentRec);
4541             dm_ICM.t_SS7_STP.FreeBookMark(CurrentRec);
4542             Exit;
4543           end;
4544
4545    dm_ICM.t_SS7_STP.GotoBookMark(CurrentRec);
4546    dm_ICM.t_SS7_STP.FreeBookMark(CurrentRec);
4547
4548    Result := CalcDistance(Lat1, Lng1,  Lat2, Lng2) / 5280.0; //RFPC0018 Calculate distance in miles
4549    WeAreWatching.LogEvent(3,'Process SS7',CLLI1+' - '+CLLI2+' distance = '+format('%9.4f .',[result]));
4550
4551  end;
4552
```

```
4553  function Tf_ss7.GetParameter(S : String) : Double;
4554  begin
4555    //RFPC0018 Finds the item 'S' in the parameter table and returns it's value
4556    //RFPC0018 If the item is not found a value of zero is returned
4557    if dm_ICM.t_SS7_Parameter.FindKey([S])
4558      then Result := dm_ICM.t_SS7_Parameter.FieldByName('Value').AsFloat
4559      else begin
4560             MessageDlg('Could not find SS7 parameter ['+ S + '].', mtError, [mbok], 0);
4561             WeAreWatching.LogEvent(1,'Process SS7','Could not find SS7 parameter ['+ S + '].');
4562             Result := 0.0;
4563           end;
4564  end;
4565
4566  function Tf_ss7.GetSS7Temp(CLLI_Code, S : String) : Boolean;
4567  begin
4568    //RFPC0018 Searches the temporary data table for item 'S' with CLLI code 'CLLI_Code'
4569    //RFPC0018 If found a logical true is return, otherwise a false
4570    Result := True;
4571    if not t_T1_SS7.FindKey([CLLI_Code, S])
4572      then begin
4573             MessageDlg('Could not find '+ S +' for '+ CLLI_Code + '.', mtError, [mbok], 0);
4574             WeAreWatching.LogEvent(1,'Process SS7','Could not find '+ S +' for '+ CLLI_Code + '.');
4575             Result := False;
4576           end;
4577  end;
4578
4579  procedure Tf_ss7.LoadH2HValues;
4580  Var
4581    Cur_Cnt    : LongInt;
4582    Rec_Cnt    : Longint;
4583    Item       : String;
4584    I          : integer;
4585    DSO        : TInvestment;
4586    AERDist    : Double;
4587    BURDist    : Double;
4588    UNGDist    : Double;
4589
4590  begin
4591    //RFPC0018 This procedure covers Section V
```

```
4592    //RFPC0018 Initialize values
4593    for I := 1 to 2 do SS7_St_IHCBDS0[I]     := 0.0;
4594    for I := 1 to 2 do SS7_St_IHDSX1DS1[I]   := 0.0;
4595    for I := 1 to 2 do SS7_St_IHDSX3DS3[I]   := 0.0;
4596    for I := 1 to 2 do SS7_St_IHFDP[I]       := 0.0;
4597    for I := 1 to 2 do SS7_St_IHADM[I]       := 0.0;
4598    for I := 1 to 2 do SS7_St_IHDSX3[I]      := 0.0;
4599    for I := 1 to 2 do SS7_St_IHDCS31[I]     := 0.0;
4600    for I := 1 to 2 do SS7_St_IHMUX31[I]     := 0.0;
4601    for I := 1 to 2 do SS7_St_IHDSX1[I]      := 0.0;
4602    for I := 1 to 2 do SS7_St_IHDCS10[I]     := 0.0;
4603    for I := 1 to 2 do SS7_St_IHCB[I]        := 0.0;
4604    for I := 1 to 2 do SS7_St_IHAERFI[I]     := 0.0;
4605    for I := 1 to 2 do SS7_St_IHBURFI[I]     := 0.0;
4606    for I := 1 to 2 do SS7_St_IHUNDFI[I]     := 0.0;
4607    for I := 1 to 2 do SS7_St_IHAERDIST[I]   := 0.0;
4608    for I := 1 to 2 do SS7_St_IHBURDIST[I]   := 0.0;
4609    for I := 1 to 2 do SS7_St_IHUNGDIST[I]   := 0.0;
4610    for I := 1 to 2 do SS7_St_IHPOLES[I]     := 0.0;
4611    for I := 1 to 2 do SS7_St_IHCOND[I]      := 0.0;
4612    for I := 1 to 2 do SS7_St_DSODistance[I]:= 0.0;
4613
4614    dm_ICM.t_Nodes.First;            //RFPC0018 Go to the beginning of the Node table
4615    while Not dm_ICM.t_Nodes.EOF do  //RFPC0018 loops through Node table
4616      begin
4617        for I := 1 to 2 do DSO[I]:= 0.0;
4618        AERDist := 0.0;
4619        BURDist := 0.0;
4620        UNGDist := 0.0;
4621
4622        //Set filter for one CLLI at a time
4623        dm_ICM.t_A_H2H.SetRange([dm_ICM.t_Nodes.FieldByName('CLLI').AsString],
4624                                [dm_ICM.t_Nodes.FieldByName('CLLI').AsString] );
4625
4626        dm_ICM.t_A_H2H.First;            //RFPC0018 Go to the beginning of the Host 2 Host table
4627        while Not dm_ICM.t_A_H2H.EOF do //RFPC0018 loops through inter-office's host to host output table
4628          begin
4629            Item := dm_ICM.t_A_H2H.FieldByName('Component').AsString;
4630            //RFPC0018 If 'Item' is one of the following sum it up
```

```
4631            if (Item = 'IHCBDS0' )
4632              then begin
4633                    SS7_St_IHCBDS0[1] := SS7_St_IHCBDS0[1] + dm_ICM.t_A_H2H.FieldByName('Total
4634  Inv').AsFloat;
4635                    SS7_St_IHCBDS0[2] := SS7_St_IHCBDS0[2] + dm_ICM.t_A_H2H.FieldByName('VS Inv').AsFloat;
4636                    DSO[1] := DSO[1] + dm_ICM.t_A_H2H.FieldByName('Total Inv').AsFloat;
4637                    DSO[2] := DSO[2] + dm_ICM.t_A_H2H.FieldByName('Total Inv').AsFloat;
4638                  end;
4639
4640            if (Item = 'IHDSX1DS1' )
4641              then begin
4642                    SS7_St_IHDSX1DS1[1] := SS7_St_IHDSX1DS1[1] + dm_ICM.t_A_H2H.FieldByName('Total
4643  Inv').AsFloat;
4644                    SS7_St_IHDSX1DS1[2] := SS7_St_IHDSX1DS1[2] + dm_ICM.t_A_H2H.FieldByName('VS
4645  Inv').AsFloat;
4646                 end;
4647
4648            if (Item = 'IHDSX3DS3' )
4649              then begin
4650                    SS7_St_IHDSX3DS3[1] := SS7_St_IHDSX3DS3[1] + dm_ICM.t_A_H2H.FieldByName('Total
4651  Inv').AsFloat;
4652                    SS7_St_IHDSX3DS3[2] := SS7_St_IHDSX3DS3[2] + dm_ICM.t_A_H2H.FieldByName('VS
4653  Inv').AsFloat;
4654                  end;
4655
4656            if (Item = 'IHFDP' )
4657              then begin
4658                    SS7_St_IHFDP[1] := SS7_St_IHFDP[1] + dm_ICM.t_A_H2H.FieldByName('Total Inv').AsFloat;
4659                    SS7_St_IHFDP[2] := SS7_St_IHFDP[2] + dm_ICM.t_A_H2H.FieldByName('VS Inv').AsFloat;
4660                  end;
4661
4662            if (Item = 'IHADM' )
4663              then begin
4664                    SS7_St_IHADM[1] := SS7_St_IHADM[1] + dm_ICM.t_A_H2H.FieldByName('Total Inv').AsFloat;
4665                    SS7_St_IHADM[2] := SS7_St_IHADM[2] + dm_ICM.t_A_H2H.FieldByName('VS Inv').AsFloat;
4666                  end;
4667
4668            if (Item = 'IHDSX3' )
4669              then begin
```

```
4670                    SS7_St_IHDSX3[1] := SS7_St_IHDSX3[1] + dm_ICM.t_A_H2H.FieldByName('Total Inv').AsFloat;
4671                    SS7_St_IHDSX3[2] := SS7_St_IHDSX3[2] + dm_ICM.t_A_H2H.FieldByName('VS Inv').AsFloat;
4672                  end;
4673
4674            if (Item = 'IHDCS31' )
4675              then begin
4676                    SS7_St_IHDCS31[1] := SS7_St_IHDCS31[1] + dm_ICM.t_A_H2H.FieldByName('Total
4677  Inv').AsFloat;
4678                    SS7_St_IHDCS31[2] := SS7_St_IHDCS31[2] + dm_ICM.t_A_H2H.FieldByName('VS Inv').AsFloat;
4679                  end;
4680
4681            if (Item = 'IHMUX31' )
4682              then begin
4683                    SS7_St_IHMUX31[1] := SS7_St_IHMUX31[1] + dm_ICM.t_A_H2H.FieldByName('Total
4684  Inv').AsFloat;
4685                    SS7_St_IHMUX31[2] := SS7_St_IHMUX31[2] + dm_ICM.t_A_H2H.FieldByName('VS Inv').AsFloat;
4686                  end;
4687
4688            if (Item = 'IHDSX1' )
4689              then begin
4690                    SS7_St_IHDSX1[1] := SS7_St_IHDSX1[1] + dm_ICM.t_A_H2H.FieldByName('Total Inv').AsFloat;
4691                    SS7_St_IHDSX1[2] := SS7_St_IHDSX1[2] + dm_ICM.t_A_H2H.FieldByName('VS Inv').AsFloat;
4692                  end;
4693
4694            if (Item = 'IHDCS10' )
4695              then begin
4696                    SS7_St_IHDCS10[1] := SS7_St_IHDCS10[1] + dm_ICM.t_A_H2H.FieldByName('Total
4697  Inv').AsFloat;
4698                    SS7_St_IHDCS10[2] := SS7_St_IHDCS10[2] + dm_ICM.t_A_H2H.FieldByName('VS Inv').AsFloat;
4699                  end;
4700
4701            if (Item = 'IHCB' )
4702              then begin
4703                    SS7_St_IHCB[1] := SS7_St_IHCB[1] + dm_ICM.t_A_H2H.FieldByName('Total Inv').AsFloat;
4704                    SS7_St_IHCB[2] := SS7_St_IHCB[2] + dm_ICM.t_A_H2H.FieldByName('VS Inv').AsFloat;
4705                  end;
4706
4707            if (Item = 'IHAERFI' )
4708              then begin
```

```
4709                    SS7_St_IHAERFI[1] := SS7_St_IHAERFI[1] + dm_ICM.t_A_H2H.FieldByName('Total
4710 Inv').AsFloat;
4711                    SS7_St_IHAERFI[2] := SS7_St_IHAERFI[2] + dm_ICM.t_A_H2H.FieldByName('VS Inv').AsFloat;
4712                  end;
4713
4714          if (Item = 'IHBURFI' )
4715            then begin
4716                    SS7_St_IHBURFI[1] := SS7_St_IHBURFI[1] + dm_ICM.t_A_H2H.FieldByName('Total
4717 Inv').AsFloat;
4718                    SS7_St_IHBURFI[2] := SS7_St_IHBURFI[2] + dm_ICM.t_A_H2H.FieldByName('VS Inv').AsFloat;
4719                  end;
4720
4721          if (Item = 'IHUNDFI' )
4722            then begin
4723                    SS7_St_IHUNDFI[1] := SS7_St_IHUNDFI[1] + dm_ICM.t_A_H2H.FieldByName('Total
4724 Inv').AsFloat;
4725                    SS7_St_IHUNDFI[2] := SS7_St_IHUNDFI[2] + dm_ICM.t_A_H2H.FieldByName('VS Inv').AsFloat;
4726                  end;
4727
4728          if (Item = 'IHAERDIST' )
4729            then begin
4730                    SS7_St_IHAERDIST[1] := SS7_St_IHAERDIST[1] + dm_ICM.t_A_H2H.FieldByName('Total
4731 Inv').AsFloat;
4732                    SS7_St_IHAERDIST[2] := SS7_St_IHAERDIST[2] + dm_ICM.t_A_H2H.FieldByName('VS
4733 Inv').AsFloat;
4734                    AERDist := dm_ICM.t_A_H2H.FieldByName('Total Inv').AsFloat;
4735                  end;
4736
4737          if (Item = 'IHBURDIST' )
4738            then begin
4739                    SS7_St_IHBURDIST[1] := SS7_St_IHBURDIST[1] + dm_ICM.t_A_H2H.FieldByName('Total
4740 Inv').AsFloat;
4741                    SS7_St_IHBURDIST[2] := SS7_St_IHBURDIST[2] + dm_ICM.t_A_H2H.FieldByName('VS
4742 Inv').AsFloat;
4743                    BURDist := dm_ICM.t_A_H2H.FieldByName('Total Inv').AsFloat;
4744                  end;
4745
4746          if (Item = 'IHUNGDIST' )
4747            then begin
```

```
4748                    SS7_St_IHUNGDIST[1] := SS7_St_IHUNGDIST[1] + dm_ICM.t_A_H2H.FieldByName('Total
4749 Inv').AsFloat;
4750                    SS7_St_IHUNGDIST[2] := SS7_St_IHUNGDIST[2] + dm_ICM.t_A_H2H.FieldByName('VS
4751 Inv').AsFloat;
4752                    UNGDist := dm_ICM.t_A_H2H.FieldByName('Total Inv').AsFloat;
4753                  end;
4754
4755          if (Item = 'IHPOLES' )
4756            then begin
4757                    SS7_St_IHPOLES[1] := SS7_St_IHPOLES[1] + dm_ICM.t_A_H2H.FieldByName('Total
4758 Inv').AsFloat;
4759                    SS7_St_IHPOLES[2] := SS7_St_IHPOLES[2] + dm_ICM.t_A_H2H.FieldByName('VS Inv').AsFloat;
4760                  end;
4761
4762          if (Item = 'IHCOND' )
4763            then begin
4764                    SS7_St_IHCOND[1] := SS7_St_IHCOND[1] + dm_ICM.t_A_H2H.FieldByName('Total Inv').AsFloat;
4765                    SS7_St_IHCOND[2] := SS7_St_IHCOND[2] + dm_ICM.t_A_H2H.FieldByName('VS Inv').AsFloat;
4766                  end;
4767
4768          if (Item = 'IHDS0' )
4769            then begin
4770                    SS7_St_IHDS0[1] := SS7_St_IHDS0[1] + dm_ICM.t_A_H2H.FieldByName('Total Inv').AsFloat;
4771                    SS7_St_IHDS0[2] := SS7_St_IHDS0[2] + dm_ICM.t_A_H2H.FieldByName('VS Inv').AsFloat;
4772                  end;
4773
4774          if (Item = 'IHDS1' )
4775            then begin
4776                    SS7_St_IHDS1[1] := SS7_St_IHDS1[1] + dm_ICM.t_A_H2H.FieldByName('Total Inv').AsFloat;
4777                    SS7_St_IHDS1[2] := SS7_St_IHDS1[2] + dm_ICM.t_A_H2H.FieldByName('VS Inv').AsFloat;
4778                    DSO[1] := DSO[1] + 24.0 * (dm_ICM.t_A_H2H.FieldByName('Total Inv').AsFloat);
4779                    DSO[2] := DSO[2] + 24.0 * (dm_ICM.t_A_H2H.FieldByName('VS Inv').AsFloat);
4780                  end;
4781
4782          if (Item = 'IHDS3' )
4783            then begin
4784                    SS7_St_IHDS3[1] := SS7_St_IHDS3[1] + dm_ICM.t_A_H2H.FieldByName('Total Inv').AsFloat;
4785                    SS7_St_IHDS3[2] := SS7_St_IHDS3[2] + dm_ICM.t_A_H2H.FieldByName('VS Inv').AsFloat;
4786                    DSO[1] := DSO[1] + 672.0 * dm_ICM.t_A_H2H.FieldByName('Total Inv').AsFloat;
```

```
4787                    DSO[2] := DSO[2] + 672.0 * dm_ICM.t_A_H2H.FieldByName('VS Inv').AsFloat;
4788                  end;
4789
4790            if (Item = 'IHSWPORTS' )
4791              then begin
4792                    SS7_St_IHSWPORTS[1] := SS7_St_IHSWPORTS[1] + dm_ICM.t_A_H2H.FieldByName('Total
4793  Inv').AsFloat;
4794                    SS7_St_IHSWPORTS[2] := SS7_St_IHSWPORTS[2] + dm_ICM.t_A_H2H.FieldByName('VS
4795  Inv').AsFloat;
4796                    DSO[1] := DSO[1] + 24.0 * dm_ICM.t_A_H2H.FieldByName('Total Inv').AsFloat;
4797                    DSO[2] := DSO[2] + 24.0 * dm_ICM.t_A_H2H.FieldByName('VS Inv').AsFloat;
4798                  end;
4799
4800            dm_ICM.t_A_H2H.Next;
4801          end;
4802
4803        SS7_St_DSODistance[1] := SS7_St_DSODistance[1] + DSO[1] * (AERDIST + BURDIST +UNGDIST);
4804        SS7_St_DSODistance[2] := SS7_St_DSODistance[2] + DSO[2] * (AERDIST + BURDIST +UNGDIST);
4805        dm_ICM.T_Nodes.Next;
4806      end;
4807  end;
4808
4809  procedure Tf_ss7.GetInterOfficeValues;
4810  Var
4811    Item       : String;
4812    I          : integer;
4813    DSO        : TInvestment;
4814    AERDist    : Double;
4815    BURDist    : Double;
4816    UNGDist    : Double;
4817
4818  begin
4819    //RFPC0018 Initialize values
4820    for I := 1 to 2 do
4821      begin
4822        IHCBDS0[I]     := 0.0;
4823        IHDSX1DS1[I]   := 0.0;
4824        IHDSX3DS3[I]   := 0.0;
4825        IHFDP[I]       := 0.0;
```

```
4826          IHADM[I]       := 0.0;
4827          IHDSX3[I]      := 0.0;
4828          IHDCS31[I]     := 0.0;
4829          IHMUX31[I]     := 0.0;
4830          IHDSX1[I]      := 0.0;
4831          IHDCS10[I]     := 0.0;
4832          IHCB[I]        := 0.0;
4833          IHAERFI[I]     := 0.0;
4834          IHBURFI[I]     := 0.0;
4835          IHUNDFI[I]     := 0.0;
4836          IHAERDIST[I]   := 0.0;
4837          IHBURDIST[I]   := 0.0;
4838          IHUNGDIST[I]   := 0.0;
4839          IHPOLES[I]     := 0.0;
4840          IHCOND[I]      := 0.0;
4841          DSODistance[i]:= 0.0;
4842          DSO[I]         := 0.0;
4843        end;
4844
4845      AERDist := 0.0;
4846      BURDist := 0.0;
4847      UNGDist := 0.0;
4848
4849      if Cur_State = State
4850        then begin
4851               //RFPC0018 Use wire-center specific
4852               dm_ICM.t_A_H2H.CancelRange;
4853               dm_ICM.t_A_H2H.SetRange([WireCtr_CLLI], [WireCtr_CLLI]);
4854
4855               while Not dm_ICM.t_A_H2H.EOF do //RFPC0018 Loop through filtered table
4856                 begin
4857                   Application.ProcessMessages;
4858                   Item := dm_ICM.t_A_H2H.FieldByName('Component').AsString;
4859                   //RFPC0018 If 'Item' is one of the following, load it
4860                   if (Item = 'IHCBDS0' )
4861                     then begin
4862                            IHCBDS0[1] := dm_ICM.t_A_H2H.FieldByName('Total Inv').AsFloat;
4863                            IHCBDS0[2] := dm_ICM.t_A_H2H.FieldByName('VS Inv').AsFloat;
4864                            DSO[1] := DSO[1] + dm_ICM.t_A_H2H.FieldByName('Total Inv').AsFloat;
```

```
4865                    DSO[2] := DSO[2] + dm_ICM.t_A_H2H.FieldByName('Total Inv').AsFloat;
4866                  end;
4867
4868              if (Item = 'IHDSX1DS1' )
4869                then begin
4870                    IHDSX1DS1[1] := dm_ICM.t_A_H2H.FieldByName('Total Inv').AsFloat;
4871                    IHDSX1DS1[2] := dm_ICM.t_A_H2H.FieldByName('VS Inv').AsFloat;
4872                  end;
4873
4874              if (Item = 'IHDSX3DS3' )
4875                then begin
4876                    IHDSX3DS3[1] := dm_ICM.t_A_H2H.FieldByName('Total Inv').AsFloat;
4877                    IHDSX3DS3[2] := dm_ICM.t_A_H2H.FieldByName('VS Inv').AsFloat;
4878                  end;
4879
4880              if (Item = 'IHFDP' )
4881                then begin
4882                    IHFDP[1] := dm_ICM.t_A_H2H.FieldByName('Total Inv').AsFloat;
4883                    IHFDP[2] := dm_ICM.t_A_H2H.FieldByName('VS Inv').AsFloat;
4884                  end;
4885
4886              if (Item = 'IHADM' )
4887                then begin
4888                    IHADM[1] := dm_ICM.t_A_H2H.FieldByName('Total Inv').AsFloat;
4889                    IHADM[2] := dm_ICM.t_A_H2H.FieldByName('VS Inv').AsFloat;
4890                  end;
4891
4892              if (Item = 'IHDSX3' )
4893                then begin
4894                    IHDSX3[1] := dm_ICM.t_A_H2H.FieldByName('Total Inv').AsFloat;
4895                    IHDSX3[2] := dm_ICM.t_A_H2H.FieldByName('VS Inv').AsFloat;
4896                  end;
4897
4898              if (Item = 'IHDCS31' )
4899                then begin
4900                    IHDCS31[1] := dm_ICM.t_A_H2H.FieldByName('Total Inv').AsFloat;
4901                    IHDCS31[2] := dm_ICM.t_A_H2H.FieldByName('VS Inv').AsFloat;
4902                  end;
4903
```

```
4904                if (Item = 'IHMUX31' )
4905                  then begin
4906                         IHMUX31[1] := dm_ICM.t_A_H2H.FieldByName('Total Inv').AsFloat;
4907                         IHMUX31[2] := dm_ICM.t_A_H2H.FieldByName('VS Inv').AsFloat;
4908                       end;
4909
4910                if (Item = 'IHDSX1' )
4911                  then begin
4912                         IHDSX1[1] := dm_ICM.t_A_H2H.FieldByName('Total Inv').AsFloat;
4913                         IHDSX1[2] := dm_ICM.t_A_H2H.FieldByName('VS Inv').AsFloat;
4914                       end;
4915
4916                if (Item = 'IHDCS10' )
4917                  then begin
4918                         IHDCS10[1] := dm_ICM.t_A_H2H.FieldByName('Total Inv').AsFloat;
4919                         IHDCS10[2] := dm_ICM.t_A_H2H.FieldByName('VS Inv').AsFloat;
4920                       end;
4921
4922                if (Item = 'IHCB' )
4923                  then begin
4924                         IHCB[1] := dm_ICM.t_A_H2H.FieldByName('Total Inv').AsFloat;
4925                         IHCB[2] := dm_ICM.t_A_H2H.FieldByName('VS Inv').AsFloat;
4926                       end;
4927
4928                if (Item = 'IHAERFI' )
4929                  then begin
4930                         IHAERFI[1] := dm_ICM.t_A_H2H.FieldByName('Total Inv').AsFloat;
4931                         IHAERFI[2] := dm_ICM.t_A_H2H.FieldByName('VS Inv').AsFloat;
4932                       end;
4933
4934                if (Item = 'IHBURFI' )
4935                  then begin
4936                         IHBURFI[1] := dm_ICM.t_A_H2H.FieldByName('Total Inv').AsFloat;
4937                         IHBURFI[2] := dm_ICM.t_A_H2H.FieldByName('VS Inv').AsFloat;
4938                       end;
4939
4940                if (Item = 'IHUNDFI' )
4941                  then begin
4942                         IHUNDFI[1] := dm_ICM.t_A_H2H.FieldByName('Total Inv').AsFloat;
```

```
4943                    IHUNDFI[2] := dm_ICM.t_A_H2H.FieldByName('VS Inv').AsFloat;
4944                  end;
4945
4946            if (Item = 'IHAERDIST' )
4947              then begin
4948                    IHAERDIST[1] := dm_ICM.t_A_H2H.FieldByName('Total Inv').AsFloat;
4949                    IHAERDIST[2] := dm_ICM.t_A_H2H.FieldByName('VS Inv').AsFloat;
4950                    AERDist := dm_ICM.t_A_H2H.FieldByName('Total Inv').AsFloat;
4951                  end;
4952
4953            if (Item = 'IHBURDIST' )
4954              then begin
4955                    IHBURDIST[1] := dm_ICM.t_A_H2H.FieldByName('Total Inv').AsFloat;
4956                    IHBURDIST[2] := dm_ICM.t_A_H2H.FieldByName('VS Inv').AsFloat;
4957                    BURDist := dm_ICM.t_A_H2H.FieldByName('Total Inv').AsFloat;
4958                  end;
4959
4960            if (Item = 'IHUNGDIST' )
4961              then begin
4962                    IHUNGDIST[1] := dm_ICM.t_A_H2H.FieldByName('Total Inv').AsFloat;
4963                    IHUNGDIST[2] := dm_ICM.t_A_H2H.FieldByName('VS Inv').AsFloat;
4964                    UNGDist := dm_ICM.t_A_H2H.FieldByName('Total Inv').AsFloat;
4965                  end;
4966
4967            if (Item = 'IHPOLES' )
4968              then begin
4969                    IHPOLES[1] := dm_ICM.t_A_H2H.FieldByName('Total Inv').AsFloat;
4970                    IHPOLES[2] := dm_ICM.t_A_H2H.FieldByName('VS Inv').AsFloat;
4971                  end;
4972
4973            if (Item = 'IHCOND' )
4974              then begin
4975                    IHCOND[1] := dm_ICM.t_A_H2H.FieldByName('Total Inv').AsFloat;
4976                    IHCOND[2] := dm_ICM.t_A_H2H.FieldByName('VS Inv').AsFloat;
4977                  end;
4978
4979            if (Item = 'IHDS0' )
4980              then begin
4981                    IHDS0[1] := dm_ICM.t_A_H2H.FieldByName('Total Inv').AsFloat;
```

```
4982                        IHDS0[2] := dm_ICM.t_A_H2H.FieldByName('VS Inv').AsFloat;
4983                      end;
4984
4985                if (Item = 'IHDS1' )
4986                  then begin
4987                        IHDS1[1] := dm_ICM.t_A_H2H.FieldByName('Total Inv').AsFloat;
4988                        IHDS1[2] := dm_ICM.t_A_H2H.FieldByName('VS Inv').AsFloat;
4989                        DSO[1] := DSO[1] + 24.0 * (dm_ICM.t_A_H2H.FieldByName('Total Inv').AsFloat);
4990                        DSO[2] := DSO[2] + 24.0 * (dm_ICM.t_A_H2H.FieldByName('VS Inv').AsFloat);
4991                      end;
4992                if (Item = 'IHDS3' )
4993                  then begin
4994                        IHDS3[1] := dm_ICM.t_A_H2H.FieldByName('Total Inv').AsFloat;
4995                        IHDS3[2] := dm_ICM.t_A_H2H.FieldByName('VS Inv').AsFloat;
4996                        DSO[1] := DSO[1] + 672.0 * dm_ICM.t_A_H2H.FieldByName('Total Inv').AsFloat;
4997                        DSO[2] := DSO[2] + 672.0 * dm_ICM.t_A_H2H.FieldByName('VS Inv').AsFloat;
4998                      end;
4999
5000                if (Item = 'IHSWPORTS' )
5001                  then begin
5002                        IHSWPORTS[1] := dm_ICM.t_A_H2H.FieldByName('Total Inv').AsFloat;
5003                        IHSWPORTS[2] := dm_ICM.t_A_H2H.FieldByName('VS Inv').AsFloat;
5004                        DSO[1] := DSO[1] + 24.0 * dm_ICM.t_A_H2H.FieldByName('Total Inv').AsFloat;
5005                        DSO[2] := DSO[2] + 24.0 * dm_ICM.t_A_H2H.FieldByName('VS Inv').AsFloat;
5006                      end;
5007
5008                dm_ICM.t_A_H2H.Next;
5009              end;
5010
5011            DSODistance[1] := DSODistance[1] + DSO[1] * (AERDIST + BURDIST +UNGDIST);
5012            DSODistance[2] := DSODistance[2] + DSO[2] * (AERDIST + BURDIST +UNGDIST);
5013            dm_ICM.t_A_H2H.CancelRange;
5014          end
5015
5016      else begin
5017            //RFPC0018 Use State values
5018            for I := 1 to 2 do IHCBDS0[I]    := SS7_St_IHCBDS0[I];
5019            for I := 1 to 2 do IHDSX1DS1[I]  := SS7_St_IHDSX1DS1[I];
5020            for I := 1 to 2 do IHDSX3DS3[I]  := SS7_St_IHDSX3DS3[I];
```

```
5021            for I := 1 to 2 do IHFDP[I]       := SS7_St_IHFDP[I];
5022            for I := 1 to 2 do IHADM[I]       := SS7_St_IHADM[I];
5023            for I := 1 to 2 do IHDSX3[I]      := SS7_St_IHDSX3[I];
5024            for I := 1 to 2 do IHDCS31[I]     := SS7_St_IHDCS31[I];
5025            for I := 1 to 2 do IHMUX31[I]     := SS7_St_IHMUX31[I];
5026            for I := 1 to 2 do IHDSX1[I]      := SS7_St_IHDSX1[I];
5027            for I := 1 to 2 do IHDCS10[I]     := SS7_St_IHDCS10[I];
5028            for I := 1 to 2 do IHCB[I]        := SS7_St_IHCB[I];
5029            for I := 1 to 2 do IHAERFI[I]     := SS7_St_IHAERFI[I];
5030            for I := 1 to 2 do IHBURFI[I]     := SS7_St_IHBURFI[I];
5031            for I := 1 to 2 do IHUNDFI[I]     := SS7_St_IHUNDFI[I];
5032            for I := 1 to 2 do IHAERDIST[I]   := SS7_St_IHAERDIST[I];
5033            for I := 1 to 2 do IHBURDIST[I]   := SS7_St_IHBURDIST[I];
5034            for I := 1 to 2 do IHUNGDIST[I]   := SS7_St_IHUNGDIST[I];
5035            for I := 1 to 2 do IHPOLES[I]     := SS7_St_IHPOLES[I];
5036            for I := 1 to 2 do IHCOND[I]      := SS7_St_IHCOND[I];
5037            for I := 1 to 2 do IHDS0[I]       := SS7_St_IHDS0[I];
5038            for I := 1 to 2 do IHDS1[I]       := SS7_St_IHDS1[I];
5039            for I := 1 to 2 do IHDS3[I]       := SS7_St_IHDS3[I];
5040            for I := 1 to 2 do IHSWPORTS[I]   := SS7_St_IHSWPORTS[I];
5041            for I := 1 to 2 do DSODistance[I]:= SS7_St_DSODistance[I];
5042          end;
5043  end;
5044
5045  procedure Tf_ss7.Process;
5046  //RFPC0018 This procedure control the SS7 processing
5047   var
5048     CLLI : String;
5049
5050   begin
5051  //RFPC0018 Check if processing is required
5052    if f_associate.GetFileName(102) = 'NO'
5053      then Exit;
5054
5055    WeAreWatching.LogEvent(1,'Process','Processed SS7.');
5056
5057  //RFPC0018 Set form control properties
5058    dbg_SS7.Visible  := False;
5059    dbn_SS7.Visible  := False;
```

```
5060    cb_Assoc.Visible := False;
5061    ts_SS7.Visible   := False;
5062    File1.Visible    := False;
5063    Query1.Visible   := False;
5064    Help1.Visible    := False;
5065
5066    p_Status.Visible := True;
5067    Width  := 310;
5068    Height := 150;
5069    show;
5070
5071    Gauge1.Progress := 0;
5072    OpenFiles(True);          //RFPC0018 Open input tables
5073    CreateActSS7;             //RFPC0018 create output table for mapping
5074    CreateTmpSS7;             //RFPC0018 create temporary data table
5075
5076    //RFPC0018 Get current state. Assume all CLLI in Node table are in the same state
5077    dm_ICM.t_Nodes.First;                                  //RFPC0018 Go to beginning of Node Table
5078    CLLI := dm_ICM.t_Nodes.FieldByName('CLLI').AsString; //RFPC0018 Get Clli Code
5079    Cur_State := Copy(CLLI, 5, 2);                         //RFPC0018 Extract state code;
5080
5081    LoadH2Hvalues;            //RFPC0018  Load Inter-Office values
5082    Gauge1.Progress := 20;
5083    SS7_Invst_STP;            //RFPC0018  Calculate STP Investments
5084    Gauge1.Progress := 40;
5085    SS7_Invst_Nodes;          //RFPC0018  Calculate SSP Investments
5086    Gauge1.Progress := 60;
5087    SS7_Invst_SCP;            //RFPC0018  Calculate SCP Investments
5088    Gauge1.Progress := 80;
5089    OutputHostInvestments;  //RFPC0018  Output Host Investments
5090    Gauge1.Progress := 90;
5091    OutputRemoteInvestments;//RFPC0018  Output Remote Investments
5092    Gauge1.Progress := 100;
5093
5094    //RFPC0018  Close all open tables
5095    dm_ICM.t_A_SS7H.Close;
5096    dm_ICM.t_A_SS7R.Close;
5097    t_T1_SS7.Close;
5098    t_T1_SS7.DeleteTable;
```

```
5099     dm_ICM.t_Switch.Close;
5100     dm_ICM.t_SwCost.Close;
5101     dm_ICM.t_Nodes.Close;
5102     dm_ICM.t_A_H2H.Close;
5103     dm_ICM.t_SS7_SSP.Close;
5104     dm_ICM.t_SS7_STP.Close;
5105     dm_ICM.t_SS7_Location.Close;
5106     dm_ICM.t_SS7_Investment.Close;
5107     dm_ICM.t_SS7_Parameter.Close;
5108     dm_ICM.t_SS7_Loading.Close;
5109
5110     f_associate.SetFileName(102, 'NO');
5111     f_associate.SetFileName(103, 'YES'); {Combine BNFs}
5112     Hide;
5113
5114     File1.Visible    := True;
5115     Query1.Visible   := True;
5116     Help1.Visible    := True;
5117     dbg_SS7.Visible  := True;
5118     dbn_SS7.Visible  := True;
5119     cb_Assoc.Visible := True;
5120     ts_SS7.Visible   := True;
5121   end;
5122
5123   procedure Tf_ss7.FormResize(Sender: TObject);
5124   //RFPC0018 responds to the form resize message
5125   begin
5126   if ClientWidth < 350
5127       then ClientWidth := 350;
5128
5129    dbg_SS7.Width := ClientWidth-20;
5130    dbn_SS7.Width := 191;
5131    ts_SS7.Width  := dbg_SS7.Width;
5132
5133    ts_SS7.Left  := 10;
5134    dbg_SS7.Left := 10;
5135    if ClientWidth > (dbn_SS7.Width + cb_Assoc.Width + 5)
5136      then dbn_SS7.Left := (ClientWidth - (dbn_SS7.Width + cb_Assoc.Width + 5)) div 2
5137      else dbn_SS7.Left := 1;
```

```
5138    cb_Assoc.Left   := dbn_SS7.Left + dbn_SS7.Width + 5;
5139    p_Status.Left   := (ClientWidth-p_Status.Width) div 2;
5140
5141    p_Status.Top := 20;
5142    dbg_SS7.Top  := l_CopyRight.Height+2;
5143    dbn_SS7.Top  := ClientHeight - (dbn_SS7.Height + 5);
5144    ts_SS7.Top   :=  dbn_SS7.Top  - (ts_SS7.Height + 5);
5145    cb_Assoc.Top :=  dbn_SS7.Top + (dbn_SS7.Height-cb_Assoc.Height) div 2;
5146
5147    dbg_SS7.Height :=  ts_SS7.Top - dbg_SS7.Top;
5148  end;
5149
5150  procedure Tf_ss7.FormShow(Sender: TObject);
5151  //RFPC0018 responds to the form show message
5152  begin
5153    cb_Assoc.Checked := False;
5154    l_Copyright.Caption := f_associate.GetFileName(5);
5155    ts_SS7.TabIndex := 0;
5156    Tab1.Caption := ts_SS7.Tabs[0]; // set help menu item
5157
5158    OpenFiles(False);
5159
5160    ds_SS7.DataSet := dm_ICM.t_SS7_SSP;          //RFPC0018 State SSP table
5161    SetCaption(0);
5162    RunFlag := False;
5163  end;
5164
5165  procedure Tf_ss7.Exit1Click(Sender: TObject);
5166  //RFPC0018 responds to the menu exit click message
5167  begin
5168   if cb_Assoc.Checked
5169    then begin
5170          f_associate.SetFileName(2010, dm_ICM.t_SS7_SSP.DatabaseName+dm_ICM.t_SS7_SSP.Tablename);
5171          f_associate.SetFileName(2017, dm_ICM.t_SS7_STP.DatabaseName+dm_ICM.t_SS7_STP.Tablename);
5172          f_associate.SetFileName(2018, dm_ICM.t_SS7_Location.DatabaseName+dm_ICM.t_SS7_Location.Tablename);
5173          f_associate.SetFileName(2019,
5174  dm_ICM.t_SS7_Investment.DatabaseName+dm_ICM.t_SS7_Investment.Tablename);
5175          f_associate.SetFileName(2020,
5176  dm_ICM.t_SS7_Parameter.DatabaseName+dm_ICM.t_SS7_Parameter.Tablename);
```

```
5177          f_associate.SetFileName(2021, dm_ICM.t_SS7_Loading.DatabaseName+dm_ICM.t_SS7_Loading.Tablename);
5178         end;
5179
5180    if RunFlag then f_associate.SetFileName(102, 'YES');
5181    cb_Assoc.Checked := False;
5182    Hide;
5183  end;
5184
5185  procedure Tf_ss7.FormHide(Sender: TObject);
5186  //RFPC0018 respondes to for hide message
5187  begin
5188    if dm_ICM.t_SS7_SSP.Active
5189      then dm_ICM.t_SS7_SSP.close;          //RFPC0018 State SSP table
5190    if dm_ICM.t_SS7_STP.Active
5191      then dm_ICM.t_SS7_STP.close;          //RFPC0018 STP table
5192    if dm_ICM.t_SS7_Location.Active
5193      then dm_ICM.t_SS7_Location.close;     //RFPC0018 STP location table
5194    if dm_ICM.t_SS7_Investment.Active
5195      then dm_ICM.t_SS7_Investment.close;   //RFPC0018 SS7 hardware and software investment table
5196    if dm_ICM.t_SS7_Parameter.Active
5197      then dm_ICM.t_SS7_Parameter.close;    //RFPC0018 SS7 network parameters table
5198    if dm_ICM.t_SS7_Loading.Active
5199      then dm_ICM.t_SS7_Loading.close;      //RFPC0018 SS7 Loading factor table
5200    if dm_ICM.t_Switch.Active
5201      then dm_ICM.t_Switch.close;           //RFPC0018 Switching table
5202    if dm_ICM.t_SwCost.Active
5203      then dm_ICM.t_SwCost.close;           //RFPC0018 Switch cost table
5204    if dm_ICM.t_Nodes.Active
5205      then dm_ICM.t_Nodes.close;            //RFPC0018 Node table
5206    if dm_ICM.t_A_H2H.Active
5207      then dm_ICM.t_A_H2H.close;            //RFPC0018 Account Host 2 Host table
5208
5209  end;
5210
5211  procedure Tf_ss7.New1Click(Sender: TObject);
5212  //RFPC0018 responds to the menu new click message
5213  begin
5214    sd_SS7.Filter := 'Database|*.db';
5215
```

```
5216    if Not sd_SS7.execute then exit;
5217    TabName := sd_SS7.FileName;
5218    cb_Assoc.Checked := True;
5219
5220    Make_SS7(ts_SS7.TabIndex);
5221    SetCaption(ts_SS7.TabIndex);
5222  end;
5223
5224  procedure Tf_ss7.Open1Click(Sender: TObject);
5225  //RFPC0018 responds to the menu open click message
5226  Var
5227   t_SS7 : TTable;
5228
5229  begin
5230    od_SS7.Filter := 'Database|*.db;*.dbf*';
5231
5232    if Not od_SS7.execute then exit;
5233    TabName := od_SS7.FileName;
5234
5235   case ts_SS7.TabIndex of
5236    0 : t_SS7 := dm_ICM.t_SS7_SSP;          //RFPC0018 State SSP table
5237    1 : t_SS7 := dm_ICM.t_SS7_STP;          //RFPC0018 STP table
5238    2 : t_SS7 := dm_ICM.t_SS7_Location;     //RFPC0018 STP location table
5239    3 : t_SS7 := dm_ICM.t_SS7_Investment;   //RFPC0018 SS7 hardware and software investment table
5240    4 : t_SS7 := dm_ICM.t_SS7_Parameter;    //RFPC0018 SS7 network parameters table
5241    5 : t_SS7 := dm_ICM.t_SS7_Loading;      //RFPC0018 SS7 Loading factor table
5242   end;
5243
5244    With t_SS7 do
5245      begin
5246         if Active then Close;
5247         DatabaseName := ExtractFilePath(TabName);
5248         TableName := ExtractFilename(TabName);
5249         TableType := ttDefault;
5250         Exclusive := True;
5251         Open;
5252      end;
5253
5254    Case ts_SS7.TabIndex of  //RFPC0018 Set properties of opened table
```

```
5255      0 : dm_ICM.t_SS7_SSP        := t_SS7;  //RFPC0018 State SSP table
5256      1 : dm_ICM.t_SS7_STP        := t_SS7;  //RFPC0018 STP table
5257      2 : dm_ICM.t_SS7_Location   := t_SS7;  //RFPC0018 STP location table
5258      3 : dm_ICM.t_SS7_Investment := t_SS7;  //RFPC0018 SS7 hardware and software investment table
5259      4 : dm_ICM.t_SS7_Parameter  := t_SS7;  //RFPC0018 SS7 network parameters table
5260      5 : dm_ICM.t_SS7_Loading    := t_SS7;  //RFPC0018 SS7 Loading factor table
5261    end;
5262
5263    SetCaption(ts_SS7.TabIndex);
5264  end;
5265
5266  procedure Tf_ss7.Import1Click(Sender: TObject);
5267  //RFPC0018 responds to the menu import click message
5268  begin
5269   case ts_SS7.TabIndex of
5270    0 : begin
5271          dm_ICM.t_SS7_SSP.close;          //RFPC0018 State SSP table
5272          ImportDataCall('', dm_ICM.t_SS7_SSP.DatabaseName+dm_ICM.t_SS7_SSP.Tablename, 10);
5273          dm_ICM.t_SS7_SSP.Open;
5274        end;
5275    1 : begin
5276          dm_ICM.t_SS7_STP.close;          //RFPC0018 STP table
5277          ImportDataCall('', dm_ICM.t_SS7_STP.DatabaseName+dm_ICM.t_SS7_STP.Tablename, 10);
5278          dm_ICM.t_SS7_STP.Open;
5279        end;
5280    2 : begin
5281          dm_ICM.t_SS7_Location.close;     //RFPC0018 STP location table
5282          ImportDataCall('', dm_ICM.t_SS7_Location.DatabaseName+dm_ICM.t_SS7_Location.Tablename, 10);
5283          dm_ICM.t_SS7_Location.Open;
5284        end;
5285    3 : begin
5286          dm_ICM.t_SS7_Investment.close;  //RFPC0018 SS7 hardware and software investment table
5287          ImportDataCall('', dm_ICM.t_SS7_Investment.DatabaseName+dm_ICM.t_SS7_Investment.Tablename, 10);
5288          dm_ICM.t_SS7_Investment.Open;
5289        end;
5290    4 : begin
5291          dm_ICM.t_SS7_Parameter.close;   //RFPC0018 SS7 network parameters table
5292          ImportDataCall('', dm_ICM.t_SS7_Parameter.DatabaseName+dm_ICM.t_SS7_Parameter.Tablename, 10);
5293          dm_ICM.t_SS7_Parameter.Open;
```

```
5294          end;
5295      5 : begin
5296            dm_ICM.t_SS7_Loading.close;      //RFPC0018 SS7 Loading factor table
5297            ImportDataCall('', dm_ICM.t_SS7_Loading.DatabaseName+dm_ICM.t_SS7_Loading.Tablename, 10);
5298            dm_ICM.t_SS7_Loading.Open;
5299          end;
5300    end;
5301  end;
5302
5303  procedure Tf_ss7.Export1Click(Sender: TObject);
5304  //RFPC0018 responds to the menu export click message
5305  begin
5306   case ts_SS7.TabIndex of
5307    0 : ExportTable(dm_ICM.t_SS7_SSP);          //RFPC0018 State SSP table
5308    1 : ExportTable(dm_ICM.t_SS7_STP);          //RFPC0018 STP table
5309    2 : ExportTable(dm_ICM.t_SS7_Location);     //RFPC0018 STP location table
5310    3 : ExportTable(dm_ICM.t_SS7_Investment);   //RFPC0018 SS7 hardware and software investment table
5311    4 : ExportTable(dm_ICM.t_SS7_Parameter);    //RFPC0018 SS7 network parameters table
5312    5 : ExportTable(dm_ICM.t_SS7_Loading);      //RFPC0018 SS7 Loading factor table
5313   end;
5314  end;
5315
5316  procedure Tf_ss7.Make_SS7(TableIndex : Integer);
5317  //RFPC0018 This procedure create empty input tables for the SS7 module
5318   Var
5319     t_SS7 : TTable;
5320
5321   begin
5322    Case TableIndex of  //RFPC0018 determine which table to make
5323     0 : t_SS7 := dm_ICM.t_SS7_SSP;           //RFPC0018 State SSP table
5324     1 : t_SS7 := dm_ICM.t_SS7_STP;           //RFPC0018 STP table
5325     2 : t_SS7 := dm_ICM.t_SS7_Location;      //RFPC0018 STP location table
5326     3 : t_SS7 := dm_ICM.t_SS7_Investment;    //RFPC0018 SS7 hardware and software investment table
5327     4 : t_SS7 := dm_ICM.t_SS7_Parameter;     //RFPC0018 SS7 network parameters table
5328     5 : t_SS7 := dm_ICM.t_SS7_Loading;       //RFPC0018 SS7 Loading factor table
5329    end;
5330
5331    with t_SS7 do
5332      begin
```

```
5333        Active := False;                              //RFPC0018 make sure is closed
5334        DatabaseName := ExtractFilePath(TabName); //RFPC0018 set database / directory
5335        TableName := ExtractFilename(TabName);    //RFPC0018 set table name
5336        TableType := ttParadox;                       //RFPC0018 set table type to paradox
5337        Exclusive := True;                            //RFPC0018 table isn't shared
5338
5339        with FieldDefs do
5340          begin
5341            Clear;
5342
5343            Case TableIndex of  //RFPC0018 add fields to table
5344              0 : begin         //RFPC0018 State SSP table
5345                    Add ('SSP_CLLI',          FTString, 11, True); //RFPC0018 SSP CLLI Code
5346                    Add ('Local_STP1',        FTString, 11, True); //RFPC0018 Local STP 1 CLLI Code
5347                    Add ('Local_STP2',        FTString, 11, True); //RFPC0018 Local STP 2 CLLI Code
5348                    Add ('ALink',             FTInteger, 0, True); //RFPC0018 Number of A Links Ports
5349                    Add ('A1_Expense',        FTFloat,   0, True); //RFPC0018 Lease Expense to Local STP1
5350                    Add ('A2_Expense',        FTFloat,   0, True); //RFPC0018 Lease Expense to Local STP1
5351                  end;
5352              1 : begin         //FPC0018 STP table
5353                    Add ('STP_CLLI',          FTString, 11, True); //RFPC0018 STP CLLI Code
5354                    Add ('WCenter_CLLI',      FTString, 11, True); //RFPC0018 Wire Center CLLI Code
5355                    Add ('State',             FTString,  2, True); //RFPC0018 State
5356                    Add ('STP_Type',          FTString,  9, True); //RFPC0018 Abbreviation of STP
5357  Vendor/Version Model
5358                    Add ('STP_Latitude',      FTFloat,   0, True); //RFPC0018 STP Latitude in degress
5359                    Add ('STP_Longitude',     FTFloat,   0, True); //RFPC0018 STP Longitude in degress
5360                    Add ('Ports_InSvc',       FTInteger, 0, True); //RFPC0018 Number of STP Ports in Service
5361                    Add ('PortCap',           FTInteger, 0, True); //RFPC0018 STP Ports Capacity
5362                    Add ('Regional_STP1',     FTString, 11, True); //RFPC0018 Local STP 1 CLLI Code
5363                    Add ('Regional_STP2',     FTString, 11, True); //RFPC0018 Local STP 2 CLLI Code
5364                    Add ('Clinks',            FTFloat,   0, True); //RFPC0018 Number of Clinks
5365                    Add ('Clink_Distance',    FTFloat,   0, True); //RFPC0018 Clink distance
5366                    Add ('Clink_Expense',     FTFloat,   0, True); //RFPC0018 Lease Expense per link to Mate
5367  STP
5368                    Add ('Dlink_Expense',     FTFloat,   0, True); //RFPC0018 Lease Expense per link to
5369  Regional STP
5370                    Add ('DB800Link_Exp',     FTFloat,   0, True); //RFPC0018 Lease Expense per link to DB800
5371  Regional STP
```

```
5372                    Add ('LIDBLink_Exp',      FTFloat,    0, True); //RFPC0018 Lease Expense per link to LIDB
5373  Regional STP
5374                    Add ('LNPLink_Exp',       FTFloat,    0, True); //RFPC0018 Lease Expense per link to LNP
5375  Regional STP
5376                    Add ('AINLink_Exp',       FTFloat,    0, True); //RFPC0018 Lease Expense per link to AIN
5377  Regional STP
5378                  end;
5379              2 : begin          //RFPC0018 STP Location table
5380                    Add ('Reg_STP1',          FTString, 11, True); //RFPC0018 Regional STP CLLI Code
5381                    Add ('DB800_PriSCP1',     FTString, 11, True); //RFPC0018 CLLI Code of STP colocated with
5382  1st pri DB800 SCP
5383                    Add ('DB800_PriSCP2',     FTString, 11, True); //RFPC0018 CLLI Code of STP colocated with
5384  2nd pri DB800 SCP
5385                    Add ('DB800_SecSCP1',     FTString, 11, True); //RFPC0018 CLLI Code of STP colocated with
5386  1st sec DB800 SCP
5387                    Add ('DB800_SecSCP2',     FTString, 11, True); //RFPC0018 CLLI Code of STP colocated with
5388  2nd sec DB800 SCP
5389                    Add ('LIDB_PriSCP1',      FTString, 11, True); //RFPC0018 CLLI Code of STP colocated with
5390  1st pri LIDB SCP
5391                    Add ('LIDB_PriSCP2',      FTString, 11, True); //RFPC0018 CLLI Code of STP colocated with
5392  2nd pri LIDB SCP
5393                    Add ('LIDB_SecSCP1',      FTString, 11, True); //RFPC0018 CLLI Code of STP colocated with
5394  1st sec LIDB SCP
5395                    Add ('LIDB_SecSCP2',      FTString, 11, True); //RFPC0018 CLLI Code of STP colocated with
5396  2nd sec LIDB SCP
5397                    Add ('LNP_PriSCP1',       FTString, 11, True); //RFPC0018 CLLI Code of STP colocated with
5398  1st pri LNP SCP
5399                    Add ('LNP_PriSCP2',       FTString, 11, True); //RFPC0018 CLLI Code of STP colocated with
5400  2nd pri LNP SCP
5401                    Add ('LNP_SecSCP1',       FTString, 11, True); //RFPC0018 CLLI Code of STP colocated with
5402  1st sec LNP SCP
5403                    Add ('LNP_SecSCP2',       FTString, 11, True); //RFPC0018 CLLI Code of STP colocated with
5404  2nd sec LNP SCP
5405                    Add ('AIN_PriSCP1',       FTString, 11, True); //RFPC0018 CLLI Code of STP colocated with
5406  1st pri AIN SCP
5407                    Add ('AIN_PriSCP2',       FTString, 11, True); //RFPC0018 CLLI Code of STP colocated with
5408  2nd pri AIN SCP
5409                    Add ('AIN_SecSCP1',       FTString, 11, True); //RFPC0018 CLLI Code of STP colocated with
5410  1st sec AIN SCP
```

```
5411                    Add ('AIN_SecSCP2',      FTString, 11, True); //RFPC0018 CLLI Code of STP colocated with
5412  2nd sec AIN SCP
5413                  end;
5414               3 : begin         //RFPC0018 SS7 hardware and software investment table
5415                    Add ('Node_Type',        FTString, 11, True); //RFPC0018 SSP/STP/SCP Type
5416                    Add ('Item',             FTString, 15, True); //RFPC0018 Investment Table Item Code
5417                    Add ('Technology',       FTString, 11, True); //RFPC0018 DMS, 5ESS, etc.
5418                    Add ('Account',          FTFloat,   0, True); //RFPC0018 Part 32 Account Code
5419                    Add ('Hardware',         FTFloat,   0, True); //RFPC0018 Hardware Investment Value
5420                    Add ('Software',         FTFloat,   0, True); //RFPC0018 Software Investment Value
5421                    Add ('Eng_Hrs',          FTFloat,   0, True); //RFPC0018 Engineering Hours
5422                    Add ('Inst_Hrs',         FTFloat,   0, True); //RFPC0018 Installation Hours
5423                    Add ('Test_Hrs',         FTFloat,   0, True); //RFPC0018 Installation Testing Hours
5424                  end;
5425               4 : begin         //RFPC0018 SS7 Network Parameters table
5426                    Add ('Code',             FTString, 15, True); //RFPC0018 Network Parameter Code
5427                    Add ('Description',      FTString, 25, True); //RFPC0018 Description Network Parameter
5428                    Add ('Value',            FTFloat,   0, True); //RFPC0018 Network Parameter Value
5429                  end;
5430               5 : begin         //RFPC0018 SS7 Loading Factor table
5431                    Add ('State',            FTString,  2, True); //RFPC0018 Two-character state abbreviation
5432                    Add ('HW_Factor',        FTFloat,   0, True); //RFPC0018 Hardware Minor Material/Supply
5433  loading factor
5434                    Add ('SW_Factor',        FTFloat,   0, True); //RFPC0018 Software Minor Material/Supply
5435  loading factor
5436                    Add ('Eng_Rate',         FTFloat,   0, True); //RFPC0018 Engineering Labor Rate per Hour
5437                    Add ('Inst_Rate',        FTFloat,   0, True); //RFPC0018 Installation Labor Rate per Hour
5438                    Add ('Test_Rate',        FTFloat,   0, True); //RFPC0018 Maintenance/Testing Labor Rate
5439  per Hour
5440                  end;
5441           end;
5442          end;
5443
5444        with IndexDefs do
5445          begin
5446            Clear;
5447             Case TableIndex of  //RFPC0018 add indices to table
5448              0 : begin         //RFPC0018 State SSP table
5449                    Add ('Main', 'SSP_CLLI', [ixPrimary, ixUnique]);
```

```
5450                end;
5451             1 : begin          //FPC0018 STP table
5452                   Add ('Main', 'STP_CLLI', [ixPrimary, ixUnique]);
5453                end;
5454             2 : begin          //RFPC0018 STP Location table
5455                   Add ('Main', 'Reg_STP1', [ixPrimary, ixUnique]);
5456                end;
5457             3 : begin          //RFPC0018 SS7 hardware and software investment table
5458                   Add ('Main', 'Node_Type;Item;Technology', [ixPrimary, ixUnique]);
5459                end;
5460             4 : begin          //RFPC0018 SS7 Network Parameters table
5461                   Add ('Main', 'Code', [ixPrimary, ixUnique]);
5462                end;
5463             5 : begin          //RFPC0018 SS7 Loading Factor table
5464                   Add ('Main', 'State', [ixPrimary, ixUnique]);
5465                end;
5466           end;
5467         end;
5468
5469       CreateTable;
5470       Open;
5471     end;
5472
5473   Case TableIndex of  //RFPC0018 Set properties of newly created table
5474    0 : dm_ICM.t_SS7_SSP        := t_SS7;  //RFPC0018 State SSP table
5475    1 : dm_ICM.t_SS7_STP        := t_SS7;  //RFPC0018 STP table
5476    2 : dm_ICM.t_SS7_Location   := t_SS7;  //RFPC0018 STP location table
5477    3 : dm_ICM.t_SS7_Investment := t_SS7;  //RFPC0018 SS7 hardware and software investment table
5478    4 : dm_ICM.t_SS7_Parameter  := t_SS7;  //RFPC0018 SS7 network parameters table
5479    5 : dm_ICM.t_SS7_Loading    := t_SS7;  //RFPC0018 SS7 Loading factor table
5480   end;
5481  end;
5482
5483  procedure Tf_ss7.CreateActSS7;
5484  //RFPC0018 This procedure create empty output tables for the SS7 module
5485  begin
5486   with dm_ICM.t_A_SS7H do
5487     begin
5488       Active := False;
```

```
5489        DatabaseName := 'modata';
5490        TableName := 'actss7H';
5491        TableType := ttParadox;
5492        Exclusive := True;
5493        with FieldDefs do
5494          begin
5495            Clear;
5496            Add ('CLLI',          FTString, 11, True);
5497            Add ('Account',       FTString, 6,  True);
5498            Add ('Component',     FTString, 16, False);
5499            Add ('Category',      FTString, 1, False);
5500            Add ('Total Inv',     FTFloat,  0,  False);
5501            Add ('VS Inv',        FTFloat,  0,  False);
5502            Add ('VIS Inv',       FTFloat,  0,  False);
5503          end;
5504
5505        with IndexDefs do
5506          begin
5507            Clear;
5508            Add('Main', 'CLLI;Account;Component', [ixPrimary, ixUnique]);
5509            Add('ByAcc', 'Account;Component', [ixCaseInsensitive]);
5510            Add('ByCat', 'Category;Account;Component', [ixCaseInsensitive]);
5511          end;
5512
5513        CreateTable;
5514        Open;
5515      end;
5516    with dm_ICM.t_A_SS7R do
5517      begin
5518        Active := False;
5519        DatabaseName := 'modata';
5520        TableName := 'actss7R';
5521        TableType := ttParadox;
5522        Exclusive := True;
5523        with FieldDefs do
5524          begin
5525            Clear;
5526            Add ('CLLI',          FTString, 11, True);
5527            Add ('Account',       FTString, 6,  True);
```

```
5528            Add ('Component',     FTString, 16, False);
5529            Add ('Category',      FTString, 1, False);
5530            Add ('Total Inv',     FTFloat,  0,  False);
5531            Add ('VS Inv',        FTFloat,  0,  False);
5532            Add ('VIS Inv',       FTFloat,  0,  False);
5533          end;
5534
5535        with IndexDefs do
5536          begin
5537            Clear;
5538            Add('Main', 'CLLI;Account;Component', [ixPrimary, ixUnique]);
5539            Add('ByAcc', 'Account;Component', [ixCaseInsensitive]);
5540            Add('ByCat', 'Category;Account;Component', [ixCaseInsensitive]);
5541          end;
5542
5543        CreateTable;
5544        Open;
5545      end;
5546  end;
5547
5548  procedure Tf_ss7.CreateTmpSS7;
5549  //RFPC0018 This procedure create the temporary data table
5550  begin
5551   with t_T1_SS7 do
5552     begin
5553       Active := False;
5554       DatabaseName := 'modata';
5555       TableName := 'TP1SS7';
5556       TableType := ttParadox;
5557       Exclusive := True;
5558       with FieldDefs do
5559         begin
5560           Clear;
5561           Add ('CLLI',          FTString, 11, True);
5562           Add ('Component',     FTString, 16, False);
5563           Add ('Total Inv',     FTFloat,  0,  False);
5564           Add ('VS Inv',        FTFloat,  0,  False);
5565           Add ('VIS Inv',       FTFloat,  0,  False);
5566         end;
```

```
5567
5568        with IndexDefs do
5569          begin
5570            Clear;
5571            Add('Main', 'CLLI;Component', [ixPrimary, ixUnique]);
5572          end;
5573
5574        CreateTable;
5575        Open;
5576      end;
5577   end;
5578
5579  procedure Tf_ss7.PrintSS7( Open : Boolean);
5580  //RFPC0018 Prints the SS7 input tables
5581  Var
5582   t_SS7   : TTable;
5583
5584  begin
5585    if Open
5586     then begin
5587            TabName :=  f_associate.GetFileName(2010); //RFPC0018 State SSP table
5588            With t_SS7 do
5589             begin
5590               if Active then Close;
5591               DatabaseName := ExtractFilePath(TabName);
5592               TableName := ExtractFilename(TabName);
5593               TableType := ttDefault;
5594               Exclusive := True;
5595               Open;
5596               GenericPrintTable(t_SS7, Nil, 0, False, 'Signaling System No.7 State SSP');
5597               Close;
5598             end;
5599
5600            TabName :=  f_associate.GetFileName(2017);  //RFPC0018 STP table
5601            With t_SS7 do
5602             begin
5603               if Active then Close;
5604               DatabaseName := ExtractFilePath(TabName);
5605               TableName := ExtractFilename(TabName);
```

```
5606              TableType := ttDefault;
5607              Exclusive := True;
5608              Open;
5609              GenericPrintTable(t_SS7, Nil, 0, False, 'Signaling System No.7 STP');
5610              Close;
5611            end;
5612
5613          TabName :=  f_associate.GetFileName(2018); //RFPC0018 STP location table
5614          With t_SS7 do
5615           begin
5616              if Active then Close;
5617              DatabaseName := ExtractFilePath(TabName);
5618              TableName := ExtractFilename(TabName);
5619              TableType := ttDefault;
5620              Exclusive := True;
5621              Open;
5622              GenericPrintTable(t_SS7, Nil, 0, False, 'Signaling System No.7 STP Location');
5623              Close;
5624           end;
5625          TabName :=  f_associate.GetFileName(2019);  //RFPC0018 SS7 hardware and software investment table
5626          With t_SS7 do
5627           begin
5628              if Active then Close;
5629              DatabaseName := ExtractFilePath(TabName);
5630              TableName := ExtractFilename(TabName);
5631              TableType := ttDefault;
5632              Exclusive := True;
5633              Open;
5634              GenericPrintTable(t_SS7, Nil, 0, False, 'Signaling System No.7 Investments');
5635              Close;
5636           end;
5637
5638          TabName :=  f_associate.GetFileName(2020); //RFPC0018 SS7 network parameters table
5639          With t_SS7 do
5640           begin
5641              if Active then Close;
5642              DatabaseName := ExtractFilePath(TabName);
5643              TableName := ExtractFilename(TabName);
5644              TableType := ttDefault;
```

```
5645              Exclusive := True;
5646              Open;
5647              GenericPrintTable(t_SS7, Nil, 0, False, 'Signaling System No.7 Network Parameters');
5648              Close;
5649             end;
5650
5651            TabName :=  f_associate.GetFileName(2021); //RFPC0018 SS7 Loading factor table
5652            With t_SS7 do
5653             begin
5654              if Active then Close;
5655              DatabaseName := ExtractFilePath(TabName);
5656              TableName := ExtractFilename(TabName);
5657              TableType := ttDefault;
5658              Exclusive := True;
5659              Open;
5660              GenericPrintTable(t_SS7, Nil, 0, False, 'Signaling System No.7 Loading Factors');
5661              Close;
5662             end;
5663          end
5664     else begin
5665            case ts_SS7.TabIndex of
5666                  //RFPC0018 SS7 Loading factor table
5667              0 : GenericPrintTable(dm_ICM.t_SS7_SSP, Nil, 0, False, 'Signaling System No.7 State SSP');
5668                  //RFPC0018 STP table
5669              1 : GenericPrintTable(dm_ICM.t_SS7_STP, Nil, 0, False, 'Signaling System No.7 STP');
5670                  //RFPC0018 STP location table
5671              2 : GenericPrintTable(dm_ICM.t_SS7_Location, Nil, 0, False, 'Signaling System No.7 STP
5672  Location');
5673                  //RFPC0018 SS7 hardware and software investment table
5674              3 : GenericPrintTable(dm_ICM.t_SS7_Investment, Nil, 0, False, 'Signaling System No.7
5675  Investments');
5676                  //RFPC0018 SS7 network parameters table
5677              4 : GenericPrintTable(dm_ICM.t_SS7_Parameter, Nil, 0, False, 'Signaling System No.7 Network
5678  Parameters');
5679                  //RFPC0018 SS7 Loading factor table
5680              5 : GenericPrintTable(dm_ICM.t_SS7_Loading, Nil, 0, False, 'Signaling System No.7 Loading
5681  Factors');
5682            end;
5683          end;
```

```
5684  end;
5685
5686  procedure Tf_ss7.Print1Click(Sender: TObject);
5687  //RFPC0018 responds to the menu print message
5688  begin
5689    if PrintDialog1.Execute then
5690      PrintSS7(False);
5691  end;
5692
5693  procedure Tf_ss7.Query1Click(Sender: TObject);
5694  //RFPC0018 respond to the menu query message
5695  begin
5696   case ts_SS7.TabIndex of
5697    0 : f_Query.Query(dm_ICM.t_SS7_SSP);          //RFPC0018 State SSP table
5698    1 : f_Query.Query(dm_ICM.t_SS7_STP);          //RFPC0018 STP table
5699    2 : f_Query.Query(dm_ICM.t_SS7_Location);     //RFPC0018 STP location table
5700    3 : f_Query.Query(dm_ICM.t_SS7_Investment);   //RFPC0018 SS7 hardware and software investment table
5701    4 : f_Query.Query(dm_ICM.t_SS7_Parameter);    //RFPC0018 SS7 network parameters table
5702    5 : f_Query.Query(dm_ICM.t_SS7_Loading);      //RFPC0018 SS7 Loading factor table
5703   end;
5704  end;
5705
5706  procedure Tf_ss7.FormActivate(Sender: TObject);
5707  //RFPC0018 reponds to the form active message
5708  begin
5709    UnitName := 'SS7';
5710  end;
5711
5712  procedure Tf_ss7.ts_SS7Change(Sender: TObject; NewTab: Integer;
5713    var AllowChange: Boolean);
5714  //RFPC0018 respond to the tabset change message
5715  begin
5716    Tab1.Caption := ts_SS7.Tabs[Newtab]; // set help menu item
5717
5718    Case NewTab of  //RFPC0018 display appropriate table
5719     0 : ds_SS7.DataSet := dm_ICM.t_SS7_SSP;          //RFPC0018 State SSP table
5720     1 : ds_SS7.DataSet := dm_ICM.t_SS7_STP;          //RFPC0018 STP table
5721     2 : ds_SS7.DataSet := dm_ICM.t_SS7_Location;     //RFPC0018 STP location table
5722     3 : ds_SS7.DataSet := dm_ICM.t_SS7_Investment;   //RFPC0018 SS7 hardware and software investment table
```

```
5723      4 : ds_SS7.DataSet := dm_ICM.t_SS7_Parameter;    //RFPC0018 SS7 network parameters table
5724      5 : ds_SS7.DataSet := dm_ICM.t_SS7_Loading;      //RFPC0018 SS7 Loading factor table
5725    end;
5726
5727    SetCaption(NewTab);
5728  end;
5729
5730  procedure Tf_ss7.Common1Click(Sender: TObject); // Help file connection
5731  begin
5732    GoHelp ('MenuIt.hlp', 305);
5733  end;
5734
5735  procedure Tf_ss7.Tab1Click(Sender: TObject); // Help file connection
5736  begin
5737     Case ts_SS7.TabIndex of
5738      0 : GoHelp ('Datatables.hlp', 441);
5739      1 : GoHelp ('Datatables.hlp', 442);
5740      2 : GoHelp ('Datatables.hlp', 443);
5741      3 : GoHelp ('Datatables.hlp', 434);
5742      4 : GoHelp ('Datatables.hlp', 439);
5743      5 : GoHelp ('Datatables.hlp', 432);
5744    end;
5745  end;
5746
5747  end.
5748
```

SWITCH.pas

```
unit Switch;
{

  This unit contains the routines to handle all of the Switching information.
  There are two tables,  switch type and switch investments.

  RFPC0017 - New fields for non Switched lines in the Node and Grid files.

     Change the current method of adding Non-switched lines to ICM. Currently
     the non switched lines are added to the business lines.  The non-switched
     lines need to be separate from the business switched lines and the residence
     switched lines.  This change would  effect the switching module and could
     effect the transport module.

     See the attachment at end of RFPC for a suggested method of separating the
     non switched lines.

     This change allows for proper costing of switching and transport which should
     be based on switched residential lines and business lines and allows for
     proper costing of the pair gain in the loop, which needs to be a combination
     of switched and non switched lines for different parts of the pair gain.


  RFPC0026 -  Protector EFI Factor Removal

    Do not apply the EFI factor to the protector material investment.
    Create an entry in the ICM Labor table to account for the per-pair labor
    investment associated with placing the protector block and protectors.
    The ICM application will need to select the labor cost from the ICM Labor
    table instead of applying the EFI factor. Apply material loadings to the
    protector investment in the material spreadsheet.


  RFPC0030
```

Augment the "Pairgain labels" arrays in "globalda.pas" to add the labels "LC4AW" and "LC4AR" to the DLCA array, labels "LC4BW" and "LC4BR" to the DLCB array, and labels "LC4CW" and "LC4CR" to the DLCC array. Expand the dimension of the arrays to a size 10.

Augment the "Pairgain labels" arrays in "grid.pas" to add the labels "PG4WW" and "PG4WR" to the DLCVar array. Expand the dimension of the array to a size 10.

Modify the "for PGT := 1 to 8 do" loop in "switch.pas" from 8 to 10.

Modify the "for PGT := 1 to 8 do" loop in "grid.pas" from 8 to 10.

NOTE: The DLC wholesale and retail spreadsheets will need to be modified to output a 4-wire investment.

This change will ensure that the ICM investment accurately reflects the cost of providing a 4-wire loop from an NGDLC pair gain device.

RFPC 0053 - Switch Investment Changes For Access Reform

These changes will allow the proper development of new rate elements in support of Access Reform and those states currently mirroring the FCC rate structure (eg., the North states). The new elements are shared end-office ports, dedicated tandem ports, dedicated end office ports and resulting "termination-less" switching. It is anticipated that these new elements will be required for the Interstate Access filing in March 2000. As such, although much of these changes are "input" changes, it is recommended that they be incorporated with the release of the new switching module.

1. The following investments are required from the switching module for both end-office and tandem switches:
   a. DS0 port (trunk termination) investments w/o network path investments
   b. Network usage investments w/o DS0 port investments
   These investments would be developed in addition to the current trunk termination (T0002) and usage investments (SWXXCS and SWXXMOU).

2. Trunk termination investments w/o network path for tandem offices needs to

be output to the subtending end offices, in addition to the End Office trunk termination w/o network path, in the ACTSWTCH and ACTCMPNT tables, as currently done for tandem switching investments.

3. The Tandem usage w/o trunk termination should also be output as separate element in the ACTSWTCH and ACTCMPNT tables for the subtending end offices.

4. Investments for tandem offices for the AMA component (SWAMA) also needs to be output for the end offices in these tables. This is similar to how the tandem usage investments are currently being done.

5. The trunk termination investments for tandem offices also needs to be output to its remote offices in the ACTSWTCH and ACTCMPNT tables. This is similar to how switch features are currently being done.

RFPC0060

The Switch Module is changed to process site specific investments as a result of CostMod and SCIS processing changes. Site specific model investment data replaces the generic switch models.

1.- XXSWINV.DB table will now include "Clli" (Number,8) field, and Fields "Switch Type" and "Size" are eliminated. The Main index of the table uses the Fields "Clli" and "Investment Type".
2.- Switch investment outputs become unique at the Clli level in the swinv.db table, and ICM switch components are built by matching the nodes.db and swinv.db tables.
3.- Substitution of the Switch type is no longer allowed and has been removed. Only allowed in Costmod and SCIS. The field substitution is removed from the switch.db and nodes.db tables.
4.- The miscellaneous features M0001,M0002,M0003,and a new M0004 for the EFI factor, are placed in the swinv.db prior to processing the switch. This is done is the GetMiscLoadings procedure. The table EFI.db is used to obtain the corresponding values, so the EFI.db table has 3 added fields "IAF", "Power Inv", and "Test Inv". The number of "line terms" is used to determine the size to be used for the EFI table.
5.- The RTU (Right to use) fees are processed in the switch, to reflect the

```
115         movement of Switch RTU fees from expense to capital. The RTU fees
116         are stored in the swinv.db table as new components. They are named
117         FRxxx, where xxx is the number of the feature to which the RTU fee
118         applies. The RTU fees are processed for each node for each feature
119         listed in feature.db, and matched according to the Switch Type.
120         The RTU fees and their account are listed on the feature.db table
121         in added fields "DMS-10","DMS-100","GTD-5", "5ESS", and "Acct".
122
123 }
124
125 interface
126
127 uses
128   SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
129   Forms, Dialogs, Menus, ExtCtrls, DBCtrls, Tabs, Grids, DBGrids, DB,
130   DBTables, StdCtrls, Printers, Globalda, Assoc, Mask, Gauges, SwSubDlg,
131   Ospmater, TablePrnt, Import, Gen_Exp, Monitr, OSPTypes;
132
133 type
134   Tf_Switching = class(TForm)
135     dbg_Switch: TDBGrid;
136     ts_Switch: TTabSet;
137     MainMenu1: TMainMenu;
138     dbn_Switch: TDBNavigator;
139     File1: TMenuItem;
140     New1: TMenuItem;
141     N1: TMenuItem;
142     Import1: TMenuItem;
143     N2: TMenuItem;
144     Exit1: TMenuItem;
145     Help1: TMenuItem;
146     cb_Assoc: TCheckBox;
147     od_Switch: TOpenDialog;
148     l_CopyRight: TLabel;
149     t_Features: TTable;
150     Open1: TMenuItem;
151     ds_switch :TDataSource;
152     sd_Switch: TSaveDialog;
153     Export1: TMenuItem;
```

```
154       p_Progress: TPanel;
155       l_Progress: TLabel;
156       Gauge1: TGauge;          // Switch Investment Table (3003)
157       t_SwAccount: TTable;
158       Print1: TMenuItem;
159       N3: TMenuItem;
160       PrintDialog1: TPrintDialog;
161       Common1: TMenuItem;
162       Tab1: TMenuItem;
163       t_Tandem: TTable;
164       procedure FormResize(Sender: TObject);
165       procedure Exit1Click(Sender: TObject);
166       procedure ts_SwitchChange(Sender: TObject; NewTab: Integer;
167         var AllowChange: Boolean);
168       procedure FormShow(Sender: TObject);
169       procedure FormHide(Sender: TObject);
170       procedure Open1Click(Sender: TObject);
171       procedure New1Click(Sender: TObject);
172       procedure Import1Click(Sender: TObject);
173       procedure Export1Click(Sender: TObject);
174       procedure Print1Click(Sender: TObject);
175       procedure FormActivate(Sender: TObject);
176       procedure Tab1Click(Sender: TObject);
177       procedure Common1Click(Sender: TObject);
178     private
179       { Private declarations }
180       procedure SetTableFilter(Tab : Integer);
181       procedure Create_ACC_Switch_Table;
182     public
183       { Public declarations }
184      procedure Entry;
185      procedure Process;
186      procedure PrintSwitch( Open : Boolean);
187      procedure PrintSwEFI( Open : Boolean);
188      procedure PrintSwCost( Open : Boolean);
189     end;
190
191   var
192     f_Switching: Tf_Switching;
```

```
193     TabName : String;            // Table name (usually from associate.getfilename)
194     Mode : Word;                 // flag indicating to open (1) or close (2) Switch processing
195     Update_cb_Investment : Boolean; // flag indicating whether to update the Investment Type data box
196
197   implementation
198    Uses nemain, Osplabor, ICMDM;
199
200   {$R *.DFM}
201   procedure Tf_Switching.Entry;
202   { This procedure is called from Nemain.pas upon selecting the Switch menu item. }
203   begin
204     ts_Switch.TabIndex := 0;     // set to indicate display the Switch Tab data
205     Mode := 1;                   // set to one to indicate entering Switch form
206     Show;                        // visually display the form
207   end; // end of Tf_Switching.Entry procedure
208
209
210   //-------------------------------------------------------------------------------------------
211
212   procedure Tf_Switching.Process;
213   { This procedure does the main Switch Processing and can be called from:
214             Nemain.pas - when Run has been selected
215             Outrpt.pas - when Output has been selected
216             Usfform.pas - when calculate has been selected.  }
217   Var
218     USG         : TUsage;        // Table of Usage Types defined in Globalda.pas
219     CLLI        : String;        // CLLI code  [Nodes Table (t_nodes)]
220     HCLLI       : String;        // Host CLLI Code
221     SwitchClli  : String;        // Clli to use when looking in the Switch invst table
222     LCLLI       : String;        // Previous CLLI
223     SW_Type     : String;        // Switch Type  [Nodes Table (t_nodes)]
224     Inv_Type    : String;        // Investment Type (Feature name)  [Feature Table (t_feature)]
225     HR          : Char;          // Host / Remote / DLC node type  [switch table (t_switch)]
226     Tier_Flag   : String;        // Labor rate category  [Nodes Table (t_nodes)]
227     LabA        : String15;      // pairgain label A
228     LabB        : String15;      // pairgain label B
229     LAdj        : Double;        // line adjustment factor
230     UAdj        : Double;        // usage adjustment factor
231     FAdj        : Double;        // feature adjustment factor
```

```
232     Loops       : Double;       // Total lines served by grid to the wire center  [Nodes Table (t_nodes)]
233     Line_Terms  : Double;       // Total # of lines served by copper feeder in grid assigned to wire center
234     EFI         : Double;       // Combined EF&I (engineered, furnished, & installed) investment [EFI table
235   (t_sw_efi)]
236     Power       : Double;       // amount of power investment adjustment applied to switch
237                                 //   investment [Switch Invest - Miscellaneous table (t_swcost)]
238     Test        : Double;       // amount of test equipment investment adjustment applied to switch
239                                 //   investment [Switch Invest - Miscellaneous table (t_swcost)]
240     MDFPROT     : Double;        //RFPC0026 Main distribution frame
241     SW_Invest   : Double;       // Total switch investment
242     Ln_Invest   : Double;       // RFPC0060 Total Lines Switch investment
243     InvFact     : double;       // amount of investment adjustment factor applied to switch
244                                 //   investment [Switch Invest - Miscellaneous table (t_swcost)]
245     R_EFI       : Double;       // Remote EFI investment
246     R_Inv       : Double;       // Remote switch investment
247     R_IF        : Double;       // Remote sensitivity factor
248     R_Pow       : Double;       // Remote Power investment
249     R_Tst       : Double;       // Remote Test Equipment investment
250     RTU01       : Double;       // Software Fees
251     Inv         : double;       // Investment for pairgain - written to actswtch table
252     VS          : double;       // Volume sensitive investment for pairgain - written to actswtch table
253     Size        : LongInt;      // Total lines served by copper feeder in grid assigned to wire center (per
254   CLLI)
255     SW_Size     : LongInt;      // Switch size (by line)  [Switch Invest table (t_swcost)]
256     Rec_Cnt     : Longint;      // Total record count of Nodes table (t_nodes) (used in Progress display)
257     Cur_Cnt     : Longint;      // Current record count in Nodes Table (t_nodes) (used in Progress display)
258     L, K        : Longint;      // L: pair gain size related to 'K'
259                                 // K: Total # of lines served by copper feeder in grid assigned to wire center
260     I, PGT      : word;         // I: general loop counter
261                                 // PGT: pairgain array for counter
262     Off_type    : word;         // Office type (class) [4-Tandem only, 4/5-Tandem/End Office,
263                                 //   5-End Office only]  [Nodes table (t_nodes)]
264     tandem_off_type :word;      // office type of tandem 4 of 4/5
265     Ok          : Boolean;      // flag to indicate if switch type was found or size > 1 [Switch table
266   (t_switch)]
267     WCBkmk      : TBookmark;    // flag record location in Nodes table (t_nodes)
268     t_Host      : tTable;       // Host-to-Host created table (3007)
269     HostFound   : Boolean;      // flag indicating if host CLLI code was found in Host-to-Host Table (t_host)
270
```

```
271  //-----------------------------------------------------------------------------------------------
272
273  //-----------------------------------------------------------------------------------------------
274
275  procedure GetSwitchType;
276  { Procedure GetSwitchType gets a switch type from the Nodes Table (t_nodes) and
277    verifies it as a valid type by searching for the same type in the switch type
278    table (t_switch).  Sets variable OK = False if NOT found. }
279   begin
280     SW_Type := dm_ICM.t_Nodes.FieldByName('Switch Type').AsString;  // get switch type from nodes table
281     if dm_ICM.t_Switch.FindKey([SW_Type]) then // search for the nodes switch type in the switch type table
282       Ok := true
283     else Ok := False;  // nodes switch type not found in switch type table, return False
284
285     if Not OK  // if switch type not found, output error message
286       then begin
287              MessageDlg(CLLI+' - Could not find switch type ['+SW_Type+']', mtError, [mbOk],0);
288              WeAreWatching.LogEvent(4,'Switch','Could not find switch type ['+SW_Type+']');
289            end;  // end 'if Not OK'
290  end;  // end GetSwitchType procedure
291
292  //-----------------------------------------------------------------------------------------------
293
294
295  function GetHost : Boolean;
296  { Function GetHost searchs the CLLI field for a CLLI which corresponds to the
297    current records Host CLLI (host clli field), in the Nodes Table,
298    and returns True or False depending on if the CLLI was found. }
299   begin
300     HCLLI    := dm_ICM.t_Nodes.FieldByName('Host CLLI').AsString; // Obtain host clli
301     WCBkMk   := dm_ICM.t_Nodes.GetBookMark; // Book mark current record
302     if Not dm_ICM.t_Nodes.FindKey([HCLLI]) // find corresponding clli using host clli
303      then begin // could not find clli
304             MessageDlg('Can not find host CLLI '+HCLLI, mterror, [mbok],0); // output message
305             GetHost := False; // return error (clli not found)
306           end  // end 'if Not t_nodes.findkey'
307      else
308        begin
309          GetSwitchType;  // get & verify nodes switch type is in switch type table
```

```
310           GetHost := True; // return found
311         end;
312       dm_ICM.t_Nodes.GotoBookMark(WCBkMk); // return to book mark
313       dm_ICM.t_Nodes.FreeBookMark(WCBkMk); // delete book mark
314
315    end; // end GetHost function
316
317   // RFPC0053
318   function HasGTETandem : boolean;
319   { Function GetGTETandem locates the GTE owned Tandem for the current Remote Clli.
320     Returns true if the GTE owned Tandem was found. This should only be called if
321     the current Clli is a remote.}
322    var TandemFound : boolean;
323    begin
324      WCBkMk  := dm_ICM.t_Nodes.GetBookMark; // Book mark current record
325      HCLLI   := dm_ICM.t_Nodes.FieldByName('Host CLLI').AsString; // Obtain host clli
326      TandemFound := false;
327      while not TandemFound and dm_ICM.t_Nodes.FindKey([HCLLI]) do
328        if (dm_ICM.t_Nodes.FieldByName('CLLI').AsString = dm_ICM.t_Nodes.FieldByName('Host CLLI').AsString)
329   then
330          TandemFound:=true
331        else HClli:=dm_ICM.t_Nodes.FieldByName('Host CLLI').AsString;
332      if TandemFound and ((dm_ICM.t_Nodes.FieldByName('Class').AsInteger = 4) or
333                          (dm_ICM.t_Nodes.FieldByName('Class').AsInteger = 45)) then
334        begin
335          Result:=true;
336          GetSwitchType;
337          tandem_off_type := dm_ICM.t_Nodes.FieldByName('Class').AsInteger;
338        end
339      else if TandemFound then
340        begin
341          MessageDlg('GTE owned tandem, '+HCLLI+', must be class 4 or 45.', mterror, [mbok],0); // output
342   message
343          Result := False; // return error (clli not found)
344        end
345      else Result:=false;
346
347      dm_ICM.t_Nodes.GotoBookMark(WCBkMk); //return to book mark
348      dm_ICM.t_Nodes.FreeBookMark(WCBkMk); //delete book mark
```

```
349
350   end; // end HasGTETandem function
351
352   //-------------------------------------------------------------------------------------------------
353
354
355   Procedure CloseTables;
356   { CloseTables procedure closes any open tables and files. }
357    begin
358      dm_ICM.t_Switch.Close;       // close switch type table
359      dm_ICM.t_Sw_EFI.Close;       // close EF&I table
360      t_SwAccount.Close;    // close created switching investment table (actswtch)
361      dm_ICM.t_SwCost.Close;       // close switch investment table
362      t_Features.Close;     // close feature list table
363      t_Tandem.Close;  //close tandems table
364      dm_ICM.t_Nodes.Close;        // close switching nodes table - wire center table
365      f_OSPMaterial.Mat('CLOSE',0);       // close outside plant material file
366      f_OSPLabor.Labor('CLOSE','XXXX');  // close outside plant labor file
367    end;    // end CloseTables procedure
368
369   //-------------------------------------------------------------------------------------------------
370
371   procedure ProcessRFPC0053;
372   { ProcessRFPC0053 procedure outputs the tandem termination and usage investments
373     for the specified Clli. }
374   var T9AvgInv, T9VSInv, T9VISInv, T2AvgInv, T2VSInv, T2VISInv,
375       ACMouAvgInv, ACMouVSInv, ACMouVISInv,
376       SW_Invest, Ln_Invest, EFI, InvFact, Power, Test, LAdj, UAdj  : double;
377     function CalcAdj:boolean;
378       begin
379         dm_ICM.t_SwCost.CancelRange; // cancel database range for T_SwCost (switch investment) table
380         // Find record in T_SwCost table for switch investment
381         if dm_ICM.t_SwCost.FindKey([HClli, 'L0001']) then
382           SW_Invest := dm_ICM.t_SwCost.FieldByName('Average Invest').AsFloat  // get investment value
383         else
384           begin  // could not find L0001 record in T_SwCost table
385             WeAreWatching.LogEvent(1,'Process','***ERROR*** Investment L0001 - for Clli: '
386                          + HClli + ' is not defined.');
387             //MessageDlg('Investment L0001 for '+ HClli +' was not found.',mtError,[mbOk],0);
```

```
388             Result:=false;
389             exit;
390           end;  // end else (could not find L0001 record)
391
392         Ln_Invest := SW_Invest; // RFPC0060 Set Total Lines switch investment
393
394         // find EFI in the switch investment table
395         if dm_ICM.t_SwCost.FindKey([HClli, 'M0004']) then
396           EFI := dm_ICM.t_SwCost.FieldByName('Average Invest').AsFloat
397         else EFI := 1.0;
398
399         { Find the investment adjustment factor in the switch investment table.
400           If found then load the investment factor, if not set factor to 0.}
401         if dm_ICM.t_SwCost.FindKey([HClli, 'M0003']) then
402            InvFact := dm_ICM.t_SwCost.FieldByName('Average Invest').AsFloat
403         else InvFact := 0.0;
404
405         { Find the power in the switch investment table. If found then
406           load the power investment factor, if not set factor to 0.}
407         if dm_ICM.t_SwCost.FindKey([HClli, 'M0001']) then
408            power := dm_ICM.t_SwCost.FieldByName('Average Invest').AsFloat
409         else power := 0.0;
410
411         { Find the test equipment in the switch investment table. If found
412           then load the test equipment investment factor, if not set factor to 0.}
413         if dm_ICM.t_SwCost.FindKey([HClli, 'M0002']) then
414            Test := dm_ICM.t_SwCost.FieldByName('Average Invest').AsFloat
415         else Test := 0.0;
416
417           SW_Invest := SW_Invest * EFI * InvFact;  // Calc total SW investment
418           if SW_Invest > 0.0
419             then begin
420                    Power := Power / SW_Invest;  // Calc power factor
421                    Test  := (Test * EFI)/SW_Invest;  // Calc test equipment factor
422                  end
423             else begin
424                    Power := 0.0;
425                    Test  := 0.0;
426                  end;
```

```
427
428           LAdj  := EFI * InvFact * (1.0 + Power + Test); // Calc Line adjustment factor
429           UAdj  := LAdj;  // Calc Usage adjustment factor
430           Result:= true;
431       end;  // CalcAdj
432   begin
433     // RFPC0053 write out the investment of the Tandem
434     if HasGTETandem then  // if the remote has a GTE owned tandem (Class 45 or 4)
435       begin
436         //GetSwitchType; // Get & verify switch type is in switch type table (t_switch)
437
438         {find record in switch invest table (t_swcost) corresponding
439          to Clli and invest type}
440         dm_ICM.t_SwCost.CancelRange; // cancel database range for T_SwCost table
441         T9AvgInv:=0;  T9VSInv:=0;  T9VISInv:=0;  T2AvgInv:=0;  T2VSInv:=0;  T2VISInv:=0;
442         if not CalcAdj then exit;
443         if dm_ICM.t_SwCost.FindKey([HClli, 'T0009']) then
444           begin
445             T9AvgInv := dm_ICM.t_SwCost.FieldByName('Average Invest').AsFloat;
446             T9VSInv  := dm_ICM.t_SwCost.FieldByName('VS Invest').AsFloat;
447             T9VISInv := T9AvgInv - T9VSInv;
448             WriteActgRecord(t_SwAccount,CLLI,'221200','TAC09','S',
449                             T9AvgInv*LAdj, T9VSInv*LAdj, T9VISInv*LAdj);
450           end;
451         if dm_ICM.t_SwCost.FindKey([HClli, 'T0002']) then
452           begin
453             T2AvgInv := dm_ICM.t_SwCost.FieldByName('Average Invest').AsFloat;
454             T2VSInv  := dm_ICM.t_SwCost.FieldByName('VS Invest').AsFloat;
455             T2VISInv := T2AvgInv - T2VSInv;
456             WriteActgRecord(t_SwAccount,CLLI,'221200','TAC02','S',
457                             T2AvgInv*LAdj, T2VSInv*LAdj, T2VISInv*LAdj);
458           end;
459         // compute minutes of use for SWACMOU
460         ACMouAvgInv := ((T2AvgInv-T9AvgInv)/28)*0.6;
461         ACMouVSInv  := ((T2VSInv-T9VSInv)/28)*0.6;
462         ACMouVISInv := ACMouAvgInv - ACMouVSInv;
463
464         // save out the usage investments
465         if (tandem_off_type = 45) then
```

```
466           begin
467             // output the switch ACAMA investments
468             if dm_ICM.t_SwCost.FindKey([HClli, 'U0000']) then
469               WriteActgRecord(t_SwAccount,CLLI,'221200','SWACAMA','S',
470                               dm_ICM.t_SwCost.FieldByName('Average Invest').AsFloat * UAdj,
471                               dm_ICM.t_SwCost.FieldByName('VS Invest').AsFloat * UAdj,
472                               (dm_ICM.t_SwCost.FieldByName('Average Invest').AsFloat -
473                                dm_ICM.t_SwCost.FieldByName('VS Invest').AsFloat) * UAdj);
474             // output the switch ACCS investments
475             if dm_ICM.t_SwCost.FindKey([HClli, 'U0007']) then
476               WriteActgRecord(t_SwAccount,CLLI,'221200','SWACCS','S',
477                               dm_ICM.t_SwCost.FieldByName('Average Invest').AsFloat * UAdj,
478                               dm_ICM.t_SwCost.FieldByName('VS Invest').AsFloat * UAdj,
479                               (dm_ICM.t_SwCost.FieldByName('Average Invest').AsFloat -
480                                dm_ICM.t_SwCost.FieldByName('VS Invest').AsFloat) * UAdj);
481             // output the switch ACMOU investments
482             if (ACMouAvgInv > 0) then
483               WriteActgRecord(t_SwAccount,CLLI,'221200','SWACMOU','S',
484                               ACMouAvgInv*UAdj,ACMouVSInv*UAdj,ACMouVISInv*UAdj)
485             else if (ACMouAvgInv = 0) or (SW_Type='GTD-5') then
486               begin
487                 if dm_ICM.t_SwCost.FindKey([HClli, 'U0008']) then
488                   WriteActgRecord(t_SwAccount,CLLI,'221200','SWACMOU','S',
489                                   dm_ICM.t_SwCost.FieldByName('Average Invest').AsFloat * UAdj,
490                                   dm_ICM.t_SwCost.FieldByName('VS Invest').AsFloat * UAdj,
491                                   (dm_ICM.t_SwCost.FieldByName('Average Invest').AsFloat -
492                                    dm_ICM.t_SwCost.FieldByName('VS Invest').AsFloat) * UAdj);
493               end
494             else {write out warning to the monitor}
495               WeAreWatching.LogEvent(1,'Process','***WARNING*** SWACMOU '+' for '+Clli+ ' is negative (T9 >
496   T2).');
497           end; // if class 45 tandem
498       end;
499     //dm_ICM.t_Nodes.GotoBookMark(WCBkMk); //return to book mark
500    // dm_ICM.t_Nodes.FreeBookMark(WCBkMk); //delete book mark
501
502     if HR = 'R' //this Node is a remote
503       then begin
504             GetHost; // Get host record
```

```
505             //GetSwitchType; // Get & verify switch type is in switch type table (t_switch)
506             //RFPC0016 Host CLLI nodes total lines to previous total lines (size)
507
508             // Output the t0002 and t0009 investments
509             dm_ICM.t_SwCost.CancelRange; // cancel database range for T_SwCost table
510             T9AvgInv:=0;  T9VSInv:=0;  T9VISInv:=0;  T2AvgInv:=0;  T2VSInv:=0;  T2VISInv:=0;
511             if not CalcAdj then exit;
512             if dm_ICM.t_SwCost.FindKey([HClli, 'T0009']) then
513               begin
514                 T9AvgInv := dm_ICM.t_SwCost.FieldByName('Average Invest').AsFloat;
515                 T9VSInv  := dm_ICM.t_SwCost.FieldByName('VS Invest').AsFloat;
516                 T9VISInv := T9AvgInv - T9VSInv;
517                 WriteActgRecord(t_SwAccount,CLLI,'221200','TACH9','S',
518                                 T9AvgInv*LAdj, T9VSInv*LAdj, T9VISInv*LAdj);
519               end;
520             if dm_ICM.t_SwCost.FindKey([HClli, 'T0002']) then
521               begin
522                 T2AvgInv := dm_ICM.t_SwCost.FieldByName('Average Invest').AsFloat;
523                 T2VSInv  := dm_ICM.t_SwCost.FieldByName('VS Invest').AsFloat;
524                 T2VISInv := T2AvgInv - T2VSInv;
525                 WriteActgRecord(t_SwAccount,CLLI,'221200','TACH2','S',
526                                 T2AvgInv*LAdj, T2VSInv*LAdj, T2VISInv*LAdj);
527               end;
528             // return bookmarks
529             //dm_ICM.t_Nodes.GotoBookMark(WCBkMk); //return to book mark
530             //dm_ICM.t_Nodes.FreeBookMark(WCBkMk); //delete book mark
531           end;  // end [if HR='R']
532
533   // end RFPC0053
534 end; // ProcessRFPC0053
535 //---------------------------------------------------------------------
536
537 procedure GetMiscLoadings;
538 { Put the miscelanous loadings M0001, M0002, M0003, M0004 in switch invst. table
539   RFPC0060}
540  var swsize,clliclass,Off_type2 :integer;
541      Clli,Clli5, offtype, temps1,temps2 :String;
542      found,found2 : boolean;
543      efi, looptest, Clli5_L0001: double;
```

```
544
545     procedure Getlineterms;
546     //RFPC0060 Get the Line terms from the Class 5 Clli that matches
547     //the Class 4/5 Clli.
548       begin
549         temps1 := copy(Clli,1,8); //get the 1st 8 characters of the 4/5 clli
550         WCBkMk := dm_ICM.t_Nodes.GetBookmark; //bookmark current Clli
551         found2 := false;
552         dm_ICM.t_Nodes.First;
553
554         while not(found2) and not(dm_ICM.t_Nodes.EOF) do
555           begin
556            //get the 1st 8 characters of this clli to compare
557            temps2 := copy(dm_ICM.t_Nodes.FieldByName('Clli').AsString,1,8);
558            //get this clli class
559            clliclass :=  dm_ICM.t_Nodes.FieldByName('Class').AsInteger;
560            if (temps1 = temps2) and (clliclass = 5) then
561              begin //if clli's 8 1st characters match then get line terms
562               swsize := dm_ICM.t_Nodes.FieldByName('Line Terms').AsInteger;
563               Clli5 := dm_ICM.t_Nodes.FieldByName('Clli').AsString;
564               found2 := true;
565              end;
566            dm_ICM.t_Nodes.Next;
567           end;
568         dm_ICM.t_Nodes.GotoBookmark(WCBkMk); //go to the current clli
569         dm_ICM.t_Nodes.FreeBookmark(WCBkMk);
570
571         if not(found2) then
572            MessageDlg('Class 5 office was not found for '+temps1,mtError,[mbOK],0);
573       end;
574   begin
575
576     dm_ICM.t_Switch.first;
577     dm_ICM.t_Nodes.First;
578     While not dm_ICM.t_Nodes.EOF do
579       begin
580         Clli := dm_ICM.t_Nodes.FieldByName('CLLI').AsString;
581         // the EFI table is based on model sizes, for each of
582         //the nodes the number of "line terms" or switched lines
```

```
583         //are used to match the next largest size of model available.
584         //If "line terms" or switched lines is greater than the
585         //largest model size, the largest model size is used.
586         //Use "line terms" for retail and switched lines for wholesale.
587
588         if Wholesale
589          then swsize := dm_ICM.t_Nodes.FieldByName('Total Lines').AsInteger -                // get Nodes
590   table Total Lines
591                        dm_ICM.t_Nodes.FieldByName('Non Switched Lines').AsInteger  // get Nodes table non
592   switched lines
593         else swsize := dm_ICM.t_Nodes.FieldByName('Line Terms').AsInteger;
594
595         //make swsize a large number so the tandems' loadings
596         //can be matched to the largest model in t_Sw_EFI
597         Off_Type2 := dm_ICM.t_Nodes.FieldByName('Class').AsInteger;
598         if (Off_Type2 = 4) then swsize:= 59999;
599
600         //if 4/5 class then get the line terms from the class five clli
601         //which matches the 1st 8 characters of the clli being processed
602         if (Off_Type2 = 45) then Getlineterms;
603
604         GetSwitchType; //get and verify nodes switch is in the switch type table
605         if OK then
606           begin
607            dm_ICM.t_Sw_EFI.CancelRange;
608            dm_ICM.t_Sw_EFI.SetRange([Sw_Type,0],[Sw_Type,100000]);
609            dm_ICM.t_Sw_EFI.First;
610            found := false;
611            while not dm_ICM.t_Sw_EFI.EOF and not found do
612              begin
613                if swsize < dm_ICM.t_Sw_EFI.FieldByName('Size').AsInteger then
614                  begin
615                    found := true;
616                  end
617                else dm_ICM.t_Sw_EFI.Next;
618              end; //end while
619            if not found then
620              begin
621                dm_ICM.t_Sw_EFI.last;
```

```
622             end;
623
624           //save the power investment into switch investment table
625           if dm_ICM.t_SwCost.Findkey([Clli,'M0001']) then
626              dm_ICM.t_SwCost.Edit
627           else dm_ICM.t_SwCost.Append;
628           dm_ICM.t_SwCost.FieldByName('Clli').AsString := Clli;
629           dm_ICM.t_SwCost.FieldByName('Investment Type').AsString := 'M0001';
630           dm_ICM.t_SwCost.FieldByName('Category').AsString := 'M';
631           dm_ICM.t_SwCost.FieldByName('Average Invest').AsFloat :=
632              dm_ICM.t_Sw_EFI.FieldByName('Power Inv').AsFloat;
633           dm_ICM.t_SwCost.Post;
634
635           //get office type
636           dm_ICM.t_Switch.FindKey([Sw_Type]);
637           offtype := dm_ICM.t_Switch.FieldByName('Host/Rem').AsString;
638
639           //save the test investment into switch investment table
640           if offtype = 'H' then
641             begin
642               if dm_ICM.t_SwCost.Findkey([Clli,'M0002']) then
643                   dm_ICM.t_SwCost.Edit
644               else dm_ICM.t_SwCost.Append;
645               dm_ICM.t_SwCost.FieldByName('Clli').AsString := Clli;
646               dm_ICM.t_SwCost.FieldByName('Investment Type').AsString := 'M0002';
647               dm_ICM.t_SwCost.FieldByName('Category').AsString := 'M';
648               dm_ICM.t_SwCost.FieldByName('Average Invest').AsFloat :=
649                 dm_ICM.t_Sw_EFI.FieldByName('Test Inv').AsFloat;
650               dm_ICM.t_SwCost.Post;
651             end;
652
653             //save the investment adjustment factor investment into switch investment table
654             if dm_ICM.t_SwCost.Findkey([Clli,'M0003']) then
655                 dm_ICM.t_SwCost.Edit
656             else dm_ICM.t_SwCost.Append;
657             dm_ICM.t_SwCost.FieldByName('Clli').AsString := Clli;
658             dm_ICM.t_SwCost.FieldByName('Investment Type').AsString := 'M0003';
659             dm_ICM.t_SwCost.FieldByName('Category').AsString := 'M';
660             dm_ICM.t_SwCost.FieldByName('Average Invest').AsFloat :=
```

```
661              dm_ICM.t_Sw_EFI.FieldByName('IAF').AsFloat;
662            dm_ICM.t_SwCost.Post;
663
664            efi := 1+ dm_ICM.t_Sw_EFI.FieldByName('EFI-I').AsFloat +
665                   dm_ICM.t_Sw_EFI.FieldByName('EFI-EF').AsFloat;
666
667            //save the EFI investment into switch investment table
668            if dm_ICM.t_SwCost.Findkey([Clli,'M0004']) then
669              dm_ICM.t_SwCost.Edit
670            else dm_ICM.t_SwCost.Append;
671            dm_ICM.t_SwCost.FieldByName('Clli').AsString := Clli;
672            dm_ICM.t_SwCost.FieldByName('Investment Type').AsString := 'M0004';
673            dm_ICM.t_SwCost.FieldByName('Category').AsString := 'M';
674            dm_ICM.t_SwCost.FieldByName('Average Invest').AsFloat := efi;
675            dm_ICM.t_SwCost.Post;
676
677            //if the clli is class 5 then write the loop test of the clli in the
678            //t_actswitch table.
679            if (Off_Type2 = 5) then
680              begin
681               looptest := efi * dm_ICM.t_Sw_EFI.FieldByName('Loop test').AsFloat;
682               WriteActgRecord(t_SwAccount,CLLI,'223200','M4TEL','L',
683                               looptest,looptest,0.0 );
684              end;
685
686            if (Off_Type2 = 45) then // if 4/5 type office get the total switch investment L0001 from
687                                     // the corresponding 5 office for which the 1st 8 chars. are the same
688              begin                  // write the L0001 investment in switch investment table under 4/5 Clli
689               if dm_ICM.t_SwCost.Findkey([Clli5,'L0001']) then
690                 Clli5_L0001 := dm_ICM.t_SwCost.FieldByName('Average Invest').AsFloat
691               else  Clli5_L0001 := 0;
692
693               if dm_ICM.t_SwCost.Findkey([Clli,'L0001']) then
694                 dm_ICM.t_SwCost.Edit
695               else dm_ICM.t_SwCost.Append;
696                 dm_ICM.t_SwCost.FieldByName('Clli').AsString := Clli;
697                 dm_ICM.t_SwCost.FieldByName('Investment Type').AsString := 'L0001';
698                 dm_ICM.t_SwCost.FieldByName('Category').AsString := 'L';
699                 dm_ICM.t_SwCost.FieldByName('Average Invest').AsFloat := Clli5_L0001;
```

```
700                  dm_ICM.t_SwCost.Post;
701
702               end;//end if 4/5 type office
703
704
705            end;  //end if OK
706          dm_ICM.t_Nodes.Next;
707        end; //end while not eof nodes table
708
709
710  end;
711
712  //---------------------------------------------------------------------
713
714  { Actual beginning of the Switching.Process procedure }
715  begin
716    { if the process switch flag is set to NO, end the switch process procedure }
717    if f_associate.GetFileName(101) <> 'YES' then exit;
718    WeAreWatching.LogEvent(1,'Process','');
719    WeAreWatching.LogEvent(1,'Process','Processing Switch');
720    Mode := 2;      // set flag to 'close' (ie done switch processing)
721    Show;           // open/display the switch form
722
723    { Open Switch Type Table }
724    TabName :=  f_associate.GetFileName(2001); // get switch type table filename and location using index
725  2001
726    With dm_ICM.t_Switch do   // set switch table variable name to t_switch and setup table
727      begin
728        if Active then Close;
729        DatabaseName := ExtractFilePath(TabName);
730        TableName := ExtractFilename(TabName);
731        TableType := ttDefault;
732        Exclusive := true;
733        IndexName := '';
734        Open;
735        CancelRange;
736      end;  // end 'with t_switch do'
737
738    { Open EFI Table }
```

```
739   TabName :=  f_associate.GetFileName(2011);  // get EFI table filename and location using index 2011
740    With dm_ICM.t_Sw_EFI do   // set Sw_EFI table variable name to t_sw_load and setup table
741      begin
742        if Active then Close;
743        DatabaseName := ExtractFilePath(TabName);
744        TableName := ExtractFilename(TabName);
745        TableType := ttDefault;
746        Exclusive := true;
747        IndexName := '';
748        Open;
749        CancelRange;
750      end;   // end 'with t_Sw_EFI do'
751
752    { Open Switch investment Table }
753    if Wholesale
754      then TabName :=  f_associate.GetFileName(3003)   // Wholesale Switch Investment table filename and
755  location using index 3003
756      else TabName :=  f_associate.GetFileName(4003);  // Retail  Switch Investment table filename and
757  location using index 4003
758
759    With dm_ICM.t_SwCost do
760      begin
761        if Active then Close;
762        DatabaseName := ExtractFilePath(TabName);
763        TableName := ExtractFilename(TabName);
764        { DONE : Change Main index of the table to Clli and Invest Type }
765        TableType := ttDefault;
766        IndexName := '';
767        Exclusive := True;
768        Open;
769        CancelRange;
770      end;   // end 'with t_swcost do'
771    Caption := 'Processing Switching [' + ExtractFilename(TabName)+']';  // set form title per table name
772
773    { Open Wire center (switching nodes) Table }
774    TabName :=  f_associate.GetFileName(2003);  // get switching nodes table filename and location using
775  index 2003
776    With dm_ICM.t_Nodes do
777      begin
```

```
778         if Active then Close;
779         DatabaseName := ExtractFilePath(TabName);
780         TableName := ExtractFilename(TabName);
781         TableType := ttDefault;
782         Exclusive := True;
783         IndexName := '';
784         Open;
785         CancelRange;
786       end;   // end 'with t_nodes do'
787
788     { Open Features List table }
789     TabName :=  f_associate.GetFileName(2007);  // get features table filename and location using index 2007
790     With t_Features do
791       begin
792         if Active then Close;
793         DatabaseName := ExtractFilePath(TabName);
794         TableName := ExtractFilename(TabName);
795         TableType := ttDefault;
796         Exclusive := True;
797         IndexName := '';
798         Open;
799         CancelRange;
800       end;   // end 'with t_features do'
801     {Open Tandems table}
802     TabName :=  f_associate.GetFileName(3011);  // get tandems table filename and location using index 3011
803     With T_Tandem do
804       begin
805         if active then close;
806         DatabaseName := ExtractFilePath(TabName);
807         TableName := ExtractFilename(TabName);
808         TableType := ttDefault;
809         Exclusive := True;
810         Indexname := '';
811         Open;
812         CancelRange;
813         First;
814       end;
815
816     Create_ACC_Switch_Table;  // Create account file for switching (actswtch)
```

```
817
818     // get the loadings EFI,Power,test, and investment adjustment factor
819     // M0001,M0002,M0003,M0004 into the switch investment table  RFPC0060
820     GetMiscLoadings;
821
822
823     dm_ICM.t_Nodes.First;  // go to first record in Nodes table
824     Rec_Cnt := dm_ICM.t_Nodes.RecordCount;  // get total record count from Nodes table
825     Cur_Cnt := 0;   // set current count to 0
826     f_OSPMaterial.Mat('OPEN',0);     // open the outside plant material file
827     f_OSPLabor.Labor('OPEN','XXXX');  // open the outside plant labor file
828
829     {iterate through the nodes table}
830     while not dm_ICM.t_Nodes.EOF do  // search through the Nodes table (wire center table)
831      begin
832         Application.ProcessMessages;  // respond to events (user actions) that have occurred
833         // determine how much progress has been made in processing switch data
834         Inc( Cur_Cnt,100 );
835         I := Cur_Cnt div Rec_Cnt;
836         if I > 100 then I := 100;
837         if Gauge1.Progress <> I then begin
838           Gauge1.Progress := I;
839           l_progress.Caption := format('%d of %d', [Cur_Cnt div 100, Rec_Cnt]);
840         end;   // end 'if gauge1.progress<>I'
841
842         CLLI       := dm_ICM.t_Nodes.FieldByName('CLLI').AsString;  // get Nodes table CLLI
843         Size       := dm_ICM.t_Nodes.FieldByName('Total Lines').AsInteger -  // get Nodes table Total Lines
844                       dm_ICM.t_Nodes.FieldByName('Non Switched Lines').AsInteger;  //RFPC0017 get Nodes table
845 non switched lines
846         Line_Terms := dm_ICM.t_Nodes.FieldByName('Line Terms').AsFloat;  // get Nodes table Line Terms
847         Off_Type   := dm_ICM.t_Nodes.FieldByName('Class').AsInteger;  // get Nodes table Office type (Class)
848
849         // set Tier_flag depending on setting of Nodes Table tier flag
850         if dm_ICM.t_Nodes.FieldByName('Tier B').AsBoolean
851          then Tier_Flag := 'B'
852          else Tier_Flag := 'A';
853
854         // write office type (class) record to 'actswtch' table
855         WriteActgRecord(t_SwAccount,CLLI,'000000','SWCLASS','S', Off_Type,Off_Type,Off_Type);
```

```
856         // write Basic Rate Interface (BRI) record to 'actswtch' table
857         WriteActgRecord(t_SwAccount,CLLI,'000000','SWBRI','S',
858                         dm_ICM.t_Nodes.FieldByName('BRI').AsFloat,
859                         dm_ICM.t_Nodes.FieldByName('BRI').AsFloat,
860                         dm_ICM.t_Nodes.FieldByName('BRI').AsFloat);
861         // write Primary Rate Interface (PRI) record to 'actswtch' table
862         WriteActgRecord(t_SwAccount,CLLI,'000000','SWPRI','S',
863                         dm_ICM.t_Nodes.FieldByName('PRI').AsFloat,
864                         dm_ICM.t_Nodes.FieldByName('PRI').AsFloat,
865                         dm_ICM.t_Nodes.FieldByName('PRI').AsFloat);
866         // write number of business lines (B1) record to 'actswtch' table
867         WriteActgRecord(t_SwAccount,CLLI,'000000','SWB1','S',
868                         dm_ICM.t_Nodes.FieldByName('B1').AsFloat,
869                         dm_ICM.t_Nodes.FieldByName('B1').AsFloat,
870                         dm_ICM.t_Nodes.FieldByName('B1').AsFloat);
871
872         Loops := Size;    // total lines
873         Ok := True;       // default OK to True (found switch type in switch type table)
874         GetSwitchType;    // get & verify nodes switch type is in switch type table
875
876         if Ok then  // is switch type found
877           begin
878             Ln_Invest := 0.0; //Initialize Total lines switch investment
879             dm_ICM.t_Switch.FindKey([SW_Type]);  // find switch type in switch type table
880             // write ID of switch type to 'actswtch' table
881             WriteActgRecord(t_SwAccount,CLLI,'000000','SWID','S',
882                             dm_ICM.t_Switch.FieldByName('ID').AsFloat,
883                             dm_ICM.t_Switch.FieldByName('ID').AsFloat,
884                             dm_ICM.t_Switch.FieldByName('ID').AsFloat);
885             // get switch type indicator for host/remote location
886             HR  := UpCase(dm_ICM.t_Switch.FieldByName('Host/Rem').AsString[1]);
887
888             case HR of     // HR: Host (H); Remote (R); Direct Loop Carrier (DLC)
889               'H' : begin  //The switch node is a Host
890                       //Off_Type    := dm_ICM.t_Nodes.FieldByName('Class').AsInteger;  // get nodes office
891   type (class)
892
893                       if dm_ICM.t_SwCost.FindKey([CLLI,'L0001']) then // if find L0001 in switch investment
894   table
```

```
895                         begin
896                           SW_Invest := dm_ICM.t_SwCost.FieldByName('Average Invest').AsFloat;  // get
897   investment value
898
899                           //RFPC0060 Set total switch investment on total lines
900                           Ln_Invest := SW_Invest; //RFPC0060 Set Total lines switch investment
901
902                           {RFPC0060 Find the EFI investment factor in the switch investment table.
903                           if found then load the investment factor, if not set factor to 1.0}
904                           if dm_ICM.t_SwCost.FindKey([Clli,'M0004']) then
905                             EFI :=  dm_ICM.t_SwCost.FieldByName('Average Invest').AsFloat
906                           else EFI := 1.0;
907
908                           { Find the investment adjustment factor in the switch investment table.
909                           if found then load the investment factor, if not set factor to 0.}
910                           if dm_ICM.t_SwCost.FindKey([Clli, 'M0003']) then
911                             InvFact := dm_ICM.t_SwCost.FieldByName('Average Invest').AsFloat
912                           else InvFact := 0.0;
913
914                           { Find the power in the switch investment table. If found then
915                           load the power investment factor, if not set factor to 0.}
916                           if dm_ICM.t_SwCost.FindKey([Clli, 'M0001']) then
917                             power := dm_ICM.t_SwCost.FieldByName('Average Invest').AsFloat
918                           else power := 0.0;
919
920                           { Find the test equipment in the switch investment table. If found
921                           then load the test equipment investment factor, if not set factor to 0.}
922                           if dm_ICM.t_SwCost.FindKey([Clli, 'M0002']) then
923                             Test := dm_ICM.t_SwCost.FieldByName('Average Invest').AsFloat
924                           else Test := 0.0;
925
926                           SW_Invest := SW_Invest *  EFI * InvFact;  // Calc total SW investment
927                           if SW_Invest > 0.0
928                             then begin
929                                    Power := Power / SW_Invest;  // Calc power factor
930                                    Test  := (Test * EFI)/SW_Invest;  // Calc test equipment factor
931                                  end
932                             else begin
933                                    Power := 0.0;
```

```
                                    Test  := 0.0;
                                  end;

                          LAdj  := EFI * InvFact * (1.0 + Power + Test); // Calc Line adjustment factor
                          UAdj  := LAdj;  // Calc Usage adjustment factor
                          FAdj  := EFI * (1.0 + Power + Test);  // Calc Feature adjustment factor

                          if Off_Type = 5 then // office type is host RFPC0060
                            begin
                            // write Main Distribution Frame investment record to 'actswtch'
                            {f_OSPMaterial.Mat(mat code, mat size) returns the unit price
                            of the specified material (code & size)}
                              WriteActgRecord(t_SwAccount,CLLI,'221200','MDF','S',
                                              f_OSPMaterial.Mat('MDF', 1)*  EFI,
                                              f_OSPMaterial.Mat('MDF', 1)*  EFI,0.0);

                            //RFPC0026 Main Distribution Frame protector investment
                              MDFPROT := f_OSPMaterial.Mat('MDFPROT', 1) +
                                      f_OSPLabor.Labor('LMDFPROT', Tier_Flag);

                            // write Main Distribution Frame Protector investment record to 'actswtch'
                              WriteActgRecord(t_SwAccount,CLLI,'242210','MDFPROT','S',
                                              MDFPROT, MDFPROT, 0.0);
                            end;  // end (off_type<>4)
                         end  // end (findkey [Clli, L0001])
                       else
                         begin // Could not find [Clli, L0001]
                           MessageDlg('Could not find Total Investment L0001 for '+CLLI+'.', mtError,
[mbOk],0);
                           WeAreWatching.LogEvent(1,'Process','***ERROR*** Could not find Total Investment
L0001 for '+CLLI+'.');
                         end;  // end else (findkey [Clli, L0001])

                    end;  // end case HR='H' (host)

               'R' : begin  // The switch node is a Remote
                       // get nodes office type (class)
                       //Off_Type  := dm_ICM.t_Nodes.FieldByName('Class').AsInteger;
```

```
973                   if dm_ICM.t_SwCost.FindKey([Clli, 'L0001']) then
974                     begin
975                       SW_Invest := dm_ICM.t_SwCost.FieldByName('Average Invest').AsFloat;
976
977                       Ln_Invest := SW_Invest; // RFPC0060 Set total lines switch investment
978
979                       {RFPC0060 Find the EFI investment factor in the switch investment table.
980                       If found then load the investment factor, if not set factor to 1.0.}
981                       if dm_ICM.t_SwCost.FindKey([Clli,'M0004']) then
982                         EFI :=  dm_ICM.t_SwCost.FieldByName('Average Invest').AsFloat
983                       else EFI := 1.0;
984
985                        // RFPC 0007 Output Main Distribution Frame investment to 'actswtch'
986                       WriteActgRecord(t_SwAccount,CLLI,'221200','MDF','S',
987                                       f_OSPMaterial.Mat('MDF', 1)*  EFI,
988                                       f_OSPMaterial.Mat('MDF', 1)*  EFI,0.0);
989
990                        //RFPC0026 Main Distribution Frame protector investment
991                       MDFPROT := f_OSPMaterial.Mat('MDFPROT', 1) +
992                                  f_OSPLabor.Labor('LMDFPROT', Tier_Flag);
993                       // write Main Distribution Frame Protector investment record to 'actswtch'
994                       WriteActgRecord(t_SwAccount,CLLI,'242210','MDFPROT','S', MDFPROT, MDFPROT, 0.0);
995
996                       { Find the investment adjustment factor in the switch investment table.
997                       If found then load the investment factor, if not set factor to 0.}
998                       if dm_ICM.t_SwCost.FindKey([Clli, 'M0003'] )then
999                         InvFact := dm_ICM.t_SwCost.FieldByName('Average Invest').AsFloat
1000                      else InvFact := 0.0;
1001
1002                      { Find the power in the switch investment table. If found then
1003                      load the power investment factor, if not set factor to 0.}
1004                      if dm_ICM.t_SwCost.FindKey([Clli, 'M0001']) then
1005                        power := dm_ICM.t_SwCost.FieldByName('Average Invest').AsFloat
1006                      else power := 0.0;
1007
1008                      { Find the test equipment in the switch investment table. If found
1009                      then load the test equipment investment factor, if not set factor to 0.}
1010                    {  if dm_ICM.t_SwCost.FindKey([Clli, 'M0002']) then
1011                        Test := dm_ICM.t_SwCost.FieldByName('Average Invest').AsFloat
```

```
1012                          else Test := 0.0; }
1013
1014                          R_EFI := EFI;              // Set remote EFI
1015                          R_Inv := SW_Invest*Size;   // Set remote Switch investment
1016                          R_IF  := InvFact;          // Set remote Sensetivity factor
1017                          R_Pow := Power;            // Set remote power
1018                         // R_Tst := Test;             // Set remote test
1019                       end  // end (if FindKey [Clli, L0001])
1020                     else
1021                        begin // Could not find [Clli, L0001]
1022                           MessageDlg('Could not find Total Investment L0001 for '+CLLI+'.', mtError,
1023   [mbOk],0);
1024                           WeAreWatching.LogEvent(1,'Process','***ERROR*** Could not find Total Investment
1025   L0001 for '+CLLI+'.');
1026                        end;  // end else (findkey [Clli, L0001])
1027
1028
1029                      if GetHost then// Get host record
1030                        begin
1031                          //GetSwitchType;  // get & verify nodes switch type is in switch type table
1032                          // get nodes office type (class)
1033                          //Off_Type   := dm_ICM.t_Nodes.FieldByName('Class').AsInteger;
1034
1035                          //dm_ICM.t_Nodes.GotoBookMark(WCBkMk); //return to book mark
1036                          //dm_ICM.t_Nodes.FreeBookMark(WCBkMk); //delete book mark
1037
1038                          if dm_ICM.t_SwCost.FindKey([HClli,'L0001']) then  // valid switch size & switch
1039   type (OK)
1040                            begin  // yes, valid
1041                              SW_Invest := dm_ICM.t_SwCost.FieldByName('Average Invest').AsFloat;
1042
1043                              { Find the EFI investment factor in the switch investment table.
1044                               If found then load the investment factor, if not set factor to 1.0.}
1045                              if dm_ICM.t_SwCost.FindKey([HClli,'M0004']) then
1046                                EFI :=  dm_ICM.t_SwCost.FieldByName('Average Invest').AsFloat
1047                              else EFI := 1.0;
1048
1049                              {RFPC0060 Find the investment adjustment factor in the switch investment table.
1050                              If found then load the investment factor, if not set factor to 0.}
```

```
1051                    if dm_ICM.t_SwCost.FindKey([HClli, 'M0003']) then
1052                       InvFact := dm_ICM.t_SwCost.FieldByName('Average Invest').AsFloat
1053                    else InvFact := 0.0;
1054
1055                    { Find the power in the switch investment table. If found then
1056                    load the power investment factor, if not set factor to 0.}
1057                    if dm_ICM.t_SwCost.FindKey([HClli, 'M0001']) then
1058                      power := dm_ICM.t_SwCost.FieldByName('Average Invest').AsFloat
1059                    else power := 0.0;
1060
1061                    { Find the test equipment in the switch investment table. If found
1062                    then load the test equipment investment factor, if not set factor to 0.}
1063                    if dm_ICM.t_SwCost.FindKey([HClli, 'M0002']) then
1064                      Test := dm_ICM.t_SwCost.FieldByName('Average Invest').AsFloat
1065                    else Test := 0.0;
1066
1067                    //Calc loaded switch investment
1068                    SW_Invest := SW_Invest * EFI * InvFact; //RFPC0016
1069
1070                    if SW_Invest > 0.0
1071                      then begin
1072                             Power := Power / SW_Invest; //RFPC0016 Calc power factor
1073                             Test  := (Test * EFI)/SW_Invest; //RFPC0016 Calc test factor
1074                           end
1075                      else begin
1076                             Power := 0.0;
1077                             Test  := 0.0;
1078                           end;
1079
1080                    //RFPC0016 Calc Line adjustment factor
1081                    LAdj  := R_EFI * R_IF * (1.0 + R_Pow/(R_Inv * R_EFI * R_IF));
1082                    UAdj  := EFI * InvFact * (1.0 + Power + Test); //Calc Usage adjustment factor
1083                    FAdj  := EFI * (1.0 + Power + Test); //Calc Feature adjustment factor
1084
1085                    EFI   := R_EFI; //set EFI to remote's EFI
1086                    SW_Invest := R_Inv * R_EFI * R_IF; //Calc remotes loaded investment
1087                  end   // end (Findkey [HClli, L0001])
1088                else
1089                  begin // Could not find [Clli, L0001]
```

```
1090                                MessageDlg('Could not find Total Investment L0001 for '+CLLI+'.', mtError,
1091  [mbOk],0);
1092                                WeAreWatching.LogEvent(1,'Process','***ERROR*** Could not find Total
1093  Investment L0001 for '+CLLI+'.');
1094                             end;  // end else (findkey [Clli, L0001])
1095
1096                         end  // end [gethost - found; (HR='R')]
1097                       else
1098                         begin //Host not found
1099                           MessageDlg('Could not find Host for '+CLLI+'.', mtError, [mbOk],0);
1100                           WeAreWatching.LogEvent(4,'Process','***ERROR*** Could not find Host for
1101  '+CLLI+'.');
1102                         end;  // end else [gethost - not found]
1103
1104                       //Off_Type   := dm_ICM.t_Nodes.FieldByName('Class').AsInteger;  // get nodes office
1105  type
1106                       end;  // end case HR='R' (remote)
1107
1108                 'D' : begin  //The switch node is a DLC (Digital Loop Carrier)
1109                         if GetHost then// Get host record
1110                           begin
1111                             //GetSwitchType;  // get & verify nodes switch type is in switch type table
1112
1113                             //dm_ICM.t_Nodes.GotoBookMark(WCBkMk); //return to book mark
1114                             //dm_ICM.t_Nodes.FreeBookMark(WCBkMk); //delete book mark
1115
1116                             if dm_ICM.t_SwCost.FindKey([HClli, 'L0001']) then  // valid switch size and
1117  switch type (OK)
1118                                begin  // yes, valid
1119                                  SW_Invest := dm_ICM.t_SwCost.FieldByName('Average Invest').AsFloat;
1120
1121                                  {RFPC0060 Find the EFI investment factor in the switch investment table.
1122                                  If found then load the investment factor, if not set factor to 1.0.}
1123                                  if dm_ICM.t_SwCost.FindKey([HClli,'M0004']) then
1124                                    EFI :=  dm_ICM.t_SwCost.FieldByName('Average Invest').AsFloat
1125                                  else EFI := 1.0;
1126
1127                                  { Find the investment adjustment factor in the switch investment table.
1128                                  If found then load the investment factor, if not set factor to 0.}
```

```
1129                                if dm_ICM.t_SwCost.FindKey([HClli, 'M0003']) then
1130                                  InvFact := dm_ICM.t_SwCost.FieldByName('Average Invest').AsFloat
1131                                else InvFact := 0.0;
1132                                                              { Find the power in the switch investment table.
1133  If found then
1134                                load the power investment factor, if not set factor to 0.}
1135                                if dm_ICM.t_SwCost.FindKey([HClli, 'M0001']) then
1136                                  power := dm_ICM.t_SwCost.FieldByName('Average Invest').AsFloat
1137                                else power := 0.0;
1138                                                              {Find the test equipment in the switch
1139  investment table. If found
1140                                then load the test equipment investment factor, if not set factor to 0.}
1141                                if dm_ICM.t_SwCost.FindKey([HClli, 'M0002']) then
1142                                  Test := dm_ICM.t_SwCost.FieldByName('Average Invest').AsFloat
1143                                else Test := 0.0;
1144
1145                                SW_Invest := SW_Invest * EFI * InvFact; // Calc total SW investment
1146                                if SW_invest > 0.0
1147                                  then begin
1148                                         Power := Power / SW_Invest; // Calc power factor
1149                                         Test  := (Test * EFI)/SW_Invest; // Calc test factor
1150                                       end
1151                                  else begin
1152                                         Power := 0.0;
1153                                         Test  := 0.0;
1154                                       end;
1155
1156                                LAdj  := EFI * InvFact * (1.0 + Power + Test); // Calc Line adjustment
1157  factor
1158                                UAdj  := LAdj; // Calc Usage adjustment factor
1159                                FAdj  := EFI * (1.0 + Power + Test); // Calc Feature adjustment factor
1160                              end  // end (Findkey [Clli, L0001])
1161                            else
1162                              begin // Could not find [Clli, L0001]
1163                                MessageDlg('Could not find Total Investment L0001 for '+CLLI+'.', mtError,
1164  [mbOk],0);
1165                                WeAreWatching.LogEvent(1,'Process','***ERROR*** Could not find Total
1166  Investment L0001 for '+CLLI+'.');
1167                              end; //
```

```
1168                          end  // end [gethost - found]
1169                        else
1170                          begin // Host not found
1171                            MessageDlg('Could not find Host for '+CLLI+'.', mtError, [mbOk],0);
1172                            WeAreWatching.LogEvent(4,'Process','***ERROR*** Could not find Host for
1173  '+CLLI+'.');
1174                          end;  // end else [host not found]
1175                      end;  // end [case HR='D']
1176            end; //End HR case
1177
1178            //RFPC0060 Check for host or tandem
1179            if (HR <> 'D') and ((Off_Type =4) or (Off_Type =5))  then
1180               begin
1181                 //RFPC0015 Ouput total switch investment on a per line basis
1182                 WriteActgRecord(t_SwAccount,CLLI,'221200','L0001','S', Ln_Invest, Ln_Invest, Ln_Invest);
1183                 //RFPC0015 Ouput total switch investment loaded
1184                 if Wholesale
1185                   then WriteActgRecord(t_SwAccount,CLLI,'221200','BM001','S', Ln_Invest * LAdj, Ln_Invest *
1186  LAdj,
1187                                        Ln_Invest * LAdj)
1188                   else WriteActgRecord(t_SwAccount,CLLI,'221200','BM002','S', Ln_Invest * LAdj, Ln_Invest *
1189  LAdj,
1190                                        Ln_Invest * LAdj);
1191               end;
1192
1193            dm_ICM.t_SwCost.CancelRange; // cancel database range for T_SwCost table
1194            t_Features.IndexFieldNames := 'Category';  // set field to search by
1195            t_Features.CancelRange;  // cancel database range for t_features table
1196            t_Features.SetRange(['T'], ['T']); // Set filter for 'Terminations' in t_features table
1197            t_Features.First;  // goto first record in table
1198
1199            while not t_Features.EOF do
1200              begin
1201                // get Investment type from t_feature table
1202                Inv_Type := t_Features.FieldByName('Feature').AsString;
1203                {find record in switch invest table (t_swcost) corresponding
1204                 to Clli & invest type}
1205                if dm_ICM.t_SwCost.FindKey([Clli, Inv_Type]) then
1206                   begin
```

```
1207                {write investment type record to 'actswtch' table using
1208                switch invest table (t_swcost) data }
1209                // RFPC 0042
1210                  if (Inv_Type='T0006') or (Inv_Type='T0007') or (Inv_Type='T0008') then
1211                    WriteActgRecord(t_SwAccount,CLLI,'221200',Inv_Type,'S',
1212                                    dm_ICM.t_SwCost.FieldByName('Average Invest').AsFloat * FAdj,
1213                                    dm_ICM.t_SwCost.FieldByName('VS Invest').AsFloat * FAdj,
1214                                    ((dm_ICM.t_SwCost.FieldByName('Average Invest').AsFloat * FAdj) -
1215                                    (dm_ICM.t_SwCost.FieldByName('VS Invest').AsFloat * FAdj)))
1216                  else
1217                    WriteActgRecord(t_SwAccount,CLLI,'221200',Inv_Type,'S',
1218                                    dm_ICM.t_SwCost.FieldByName('Average Invest').AsFloat * LAdj,
1219                                    dm_ICM.t_SwCost.FieldByName('VS Invest').AsFloat * LAdj,
1220                                    ((dm_ICM.t_SwCost.FieldByName('Average Invest').AsFloat * LAdj) -
1221                                    (dm_ICM.t_SwCost.FieldByName('VS Invest').AsFloat * LAdj)));
1222                end;  // end [if t_swcost.findkey]
1223
1224              t_Features.Next;   // get next record in t_features table
1225            end;  // end [while not t_features.eof]
1226
1227          if Off_Type = 5 then //RFPC0060 process only if End Office
1228          ProcessRFPC0053;  // RFPC0053 output the tandem termination and usage investments
1229
1230          if HR = 'R' then//this Node is a remote
1231            begin
1232              GetHost; // Get host record
1233              //GetSwitchType; // Get & verify switch type is in switch type table (t_switch)
1234
1235              //dm_ICM.t_Nodes.GotoBookMark(WCBkMk); //return to book mark
1236              //dm_ICM.t_Nodes.FreeBookMark(WCBkMk); //delete book mark
1237            end;  // end [if HR='R']
1238
1239          if HR ='D' then //This NODE is a DLC (Digital Loop Carrier)
1240            begin
1241              // Get DLC investments
1242              // load the number of loop which terminate on the DLC
1243              k := dm_ICM.t_Nodes.FieldByName('Line Terms').AsInteger;
1244
1245              for PGT := 1 to 10 do   //RFPC0030 run loop for 10 iterations
```

```
1246                  begin
1247                    // get the material size using pairgain label & nodes line term
1248                    L := round(f_OSPMaterial.FindSize(DLCA[PGT], K));
1249                    if L = 0 then
1250                       begin
1251                        MessageDlg('Material ' + DLCA[PGT] +' not found.', mtError, [mbok], 0);
1252                                  WeAreWatching.LogEvent(1,'Process Feeder','***ERROR*** Material
1253  '+DLCA[PGT]+' not found.');
1254                       end;
1255                    LabA := DLCA[PGT] + InttoStr(L);  // label pairgain A with size
1256                    LabB := DLCB[PGT] + InttoStr(L);  // label pairgain B with size
1257
1258                    // calculate investment for label A
1259                    Inv  :=  f_OSPMaterial.Mat(DLCA[PGT], L) + f_OSPLabor.Labor(LabA,  Tier_Flag);
1260                    // calculate volume sensitive for label A
1261                    VS   := (f_OSPMaterial.Mat(DLCA[PGT], L) + f_OSPLabor.Labor(LabA,  Tier_Flag))/L*K;
1262                    // write pairgain (A) data to 'actswtch' table
1263                    WriteActgRecord(t_SwAccount,CLLI,'223200','SW'+DLCA[PGT],'S', Inv, VS, Inv-VS);
1264                    // calculate investment for label B
1265                    Inv  :=  f_OSPMaterial.Mat(DLCB[PGT], L) + f_OSPLabor.Labor(LabB,  Tier_Flag);
1266                    // write pairgain (B) data to 'actswtch' table
1267                    WriteActgRecord(t_SwAccount,CLLI,'223200','SW'+DLCB[PGT],'S', Inv, Inv, 0.0);
1268
1269                    //RFPC0029 Output DLC invest on a per line basis
1270                    // Part B was the last calculate value, so part A needs to be added for
1271                    // the total investment of the DLC
1272                    Inv  :=  Inv + f_OSPMaterial.Mat(DLCA[PGT], L) + f_OSPLabor.Labor(LabA,  Tier_Flag);
1273
1274                    if Pos('LCBAW', LabA) = 1 then  // wholesale
1275                       WriteActgRecord(t_SwAccount,CLLI,'223200','BM001','S', Inv, Inv, Inv);
1276
1277                    if Pos('LCBAR', LabA) = 1  then // retail
1278                       WriteActgRecord(t_SwAccount,CLLI,'223200','BM002','S', Inv, Inv, Inv);
1279
1280                  end;  // end [for PGT]
1281              end;  // end [if HR='D']
1282
1283            if (HR = 'R') or (HR = 'D') then
1284              SwitchClli := HClli
```

```
1285            else SwitchClli := Clli;
1286
1287            dm_ICM.t_SwCost.CancelRange; // cancel database range for T_SwCost table
1288            t_Features.IndexFieldNames := 'Category';  // set field to search by
1289            if Off_Type = 5 then // office type is not tandem
1290               begin
1291                 GetSwitchType;
1292                 t_Features.CancelRange;  // cancel t_features database range
1293
1294                 if (HR = 'R')        //RFPC0060 If Remote, then copy terminations
1295                   then begin
1296                          if dm_ICM.t_SwCost.FindKey([SwitchClli, 'T0006']) then //RFPC0060
1297                            begin
1298                              // write investment type record to 'actswtch' using switch investment data
1299                              WriteActgRecord(t_SwAccount,CLLI,'221200','T0006','S',
1300  //RFPC0060
1301                                              dm_ICM.t_SwCost.FieldByName('Average Invest').AsFloat * Fadj,
1302  //RFPC0060
1303                                              dm_ICM.t_SwCost.FieldByName('VS Invest').AsFloat * Fadj,
1304  //RFPC0060
1305                                              ((dm_ICM.t_SwCost.FieldByName('Average Invest').AsFloat * Fadj) -
1306  //RFPC0060
1307                                              (dm_ICM.t_SwCost.FieldByName('VS Invest').AsFloat * Fadj)));
1308  //RFPC0060
1309                            end;
1310                        end;
1311
1312                 if (HR = 'R') or (HR = 'D')        //RFPC0060 If Remote, then copy terminations
1313                   then begin
1314                          if dm_ICM.t_SwCost.FindKey([SwitchClli, 'T0003']) then //RFPC0060
1315                            begin
1316                              // write investment type record to 'actswtch' using switch investment data
1317                              WriteActgRecord(t_SwAccount,CLLI,'221200','T0003','S',
1318  //RFPC0060
1319                                              dm_ICM.t_SwCost.FieldByName('Average Invest').AsFloat * Fadj,
1320  //RFPC0060
1321                                              dm_ICM.t_SwCost.FieldByName('VS Invest').AsFloat * Fadj,
1322  //RFPC0060
```

```
1323                                    ((dm_ICM.t_SwCost.FieldByName('Average Invest').AsFloat * Fadj) -
1324  //RFPC0060
1325                                    (dm_ICM.t_SwCost.FieldByName('VS Invest').AsFloat * Fadj)));
1326  //RFPC0060
1327                          end;
1328                        if dm_ICM.t_SwCost.FindKey([SwitchClli, 'T0007']) then //RFPC0060
1329                          begin
1330                            // write investment type record to 'actswtch' using switch investment data
1331                            WriteActgRecord(t_SwAccount,CLLI,'221200','T0007','S',
1332  //RFPC0060
1333                                    dm_ICM.t_SwCost.FieldByName('Average Invest').AsFloat * Fadj,
1334  //RFPC0060
1335                                    dm_ICM.t_SwCost.FieldByName('VS Invest').AsFloat * Fadj,
1336  //RFPC0060
1337                                    ((dm_ICM.t_SwCost.FieldByName('Average Invest').AsFloat * Fadj) -
1338  //RFPC0060
1339                                    (dm_ICM.t_SwCost.FieldByName('VS Invest').AsFloat * Fadj)));
1340  //RFPC0060
1341                          end;
1342                        if dm_ICM.t_SwCost.FindKey([SwitchClli, 'T0008']) then //RFPC0060
1343                          begin
1344                            // write investment type record to 'actswtch' using switch investment data
1345                            WriteActgRecord(t_SwAccount,CLLI,'221200','T0008','S',
1346  //RFPC0060
1347                                    dm_ICM.t_SwCost.FieldByName('Average Invest').AsFloat * Fadj,
1348  //RFPC0060
1349                                    dm_ICM.t_SwCost.FieldByName('VS Invest').AsFloat * Fadj,
1350  //RFPC0060
1351                                    ((dm_ICM.t_SwCost.FieldByName('Average Invest').AsFloat * Fadj) -
1352  //RFPC0060
1353                                    (dm_ICM.t_SwCost.FieldByName('VS Invest').AsFloat * Fadj)));
1354  //RFPC0060
1355                          end;
1356                      end;
1357
1358                RTU01 := 0.0;  //Initialize total software fees
1359                t_Features.SetRange(['F'], ['F']);  // set filter for 'Feature' in t_features table
1360                t_Features.First;  // go to first record in the t_features table
1361                while not t_Features.EOF do
```

```
1362                    begin
1363                    // get investment type from t_features table
1364                      Inv_Type := t_Features.FieldByName('Feature').AsString;
1365                      {find record in switch investment table (t_swcost) corresponding
1366                      to switch type, invest type, & switch size}
1367                      if dm_ICM.t_SwCost.FindKey([SwitchClli, Inv_Type]) then
1368                         begin
1369                         // write investment type record to 'actswtch' using switch investment data
1370                           WriteActgRecord(t_SwAccount,CLLI,'221200',Inv_Type,'S',
1371                                           dm_ICM.t_SwCost.FieldByName('Average Invest').AsFloat * Fadj,
1372                                           dm_ICM.t_SwCost.FieldByName('VS Invest').AsFloat * Fadj,
1373                                           ((dm_ICM.t_SwCost.FieldByName('Average Invest').AsFloat * Fadj) -
1374                                           (dm_ICM.t_SwCost.FieldByName('VS Invest').AsFloat * Fadj)));
1375                         end;  // end [if t_swcost.findkey]
1376
1377                         {Write the RTU fees to the actswitch table RFPC 060}
1378                         // The RTU fee is matched according to the node's
1379                         // switch type.
1380                         Inv_Type := 'R' + copy(Inv_Type,2,4);
1381                         if dm_ICM.t_SwCost.FindKey([SwitchClli, Inv_Type])
1382                           then begin
1383                                  WriteActgRecord(t_SwAccount,CLLI, '269030', 'F'+Inv_Type, 'S',
1384                                                  dm_ICM.t_SwCost.FieldByName('Average Invest').AsFloat,
1385                                                  dm_ICM.t_SwCost.FieldByName('Average Invest').AsFloat,
1386                                                  0.0);
1387
1388                                 // Total Software Fees
1389                                 if t_Features.FieldByName('Group').AsBoolean
1390                                   then begin
1391                                          if HR = 'H' then RTU01 := RTU01 + // if a feature is a group, add it
1392  only to a host
1393                                                                  dm_ICM.t_SwCost.FieldByName('Average
1394  Invest').AsFloat;
1395                                      end
1396                                   else RTU01 := RTU01 + dm_ICM.t_SwCost.FieldByName('Average
1397  Invest').AsFloat * Loops;
1398                               end;
1399                      t_Features.Next;  // get next record in t_features table
1400                    end;  // end [while not t_features.eof]
```

```
1401                WriteActgRecord(t_SwAccount,CLLI, '269030', 'RTU01', 'S', RTU01, RTU01, 0.0);
1402             end;  // end [if off_type=5]
1403
1404          t_Features.CancelRange;  // cancel t_features search ranges
1405          t_Features.SetRange(['U'], ['U']);  // set filters for 'Usage' in t_features table
1406          t_Features.First;  // go to first t_features record
1407
1408          // set default values to 'Usage' features
1409          USG.Line_Line_Call     := 0.0;
1410          USG.Line_Line_CCS      := 0.0;
1411          USG.Line_Trunk_Call    := 0.0;
1412          USG.Line_Trunk_CCS     := 0.0;
1413          USG.Trunk_Line_Call    := 0.0;
1414          USG.Trunk_Line_CCS     := 0.0;
1415          USG.Trunk_Trunk_Call   := 0.0;
1416          USG.Trunk_Trunk_CCS    := 0.0;
1417          USG.AMA_Cost_Call      := 0.0;
1418          USG.Line_Line_CallVS   := 0.0;
1419          USG.Line_Line_CCSVS    := 0.0;
1420          USG.Line_Trunk_CallVS  := 0.0;
1421          USG.Line_Trunk_CCSVS   := 0.0;
1422          USG.Trunk_Line_CallVS  := 0.0;
1423          USG.Trunk_Line_CCSVS   := 0.0;
1424          USG.Trunk_Trunk_CallVS := 0.0;
1425          USG.Trunk_Trunk_CCSVS  := 0.0;
1426          USG.AMA_Cost_CallVS    := 0.0;
1427
1428          {find record in switch invest table (t_swcost) corresponding to
1429          switch type, usage feature, and switch size}
1430          if dm_ICM.t_SwCost.FindKey([SwitchClli, 'U0000']) then
1431            begin
1432              // calculate Usage investment (U0000)
1433              USG.AMA_Cost_Call := dm_ICM.t_SwCost.FieldByName('Average Invest').AsFloat * UAdj;
1434              // calculate Usage volume sensitive investment (U0000)
1435              USG.AMA_Cost_CallVS:= dm_ICM.t_SwCost.FieldByName('VS Invest').AsFloat * UAdj;
1436            end;  // end [if t_swcost.findkey(U0000)]
1437          if dm_ICM.t_SwCost.FindKey([SwitchClli, 'U0001']) then
1438            begin
1439              // calculate Usage investment (U0001)
```

```
1440              USG.Line_Line_Call  := dm_ICM.t_SwCost.FieldByName('Average Invest').AsFloat * UAdj;
1441              // calculate Usage volume sensitive investment (U0001)
1442              USG.Line_Line_CallVS:= dm_ICM.t_SwCost.FieldByName('VS Invest').AsFloat * UAdj;
1443            end;  // end [if t_swcost.findkey(U0001)]
1444
1445          if dm_ICM.t_SwCost.FindKey([SwitchClli, 'U0002']) then
1446            begin
1447              // calculate Usage investment (U0002)
1448              USG.Line_Line_CCS  := dm_ICM.t_SwCost.FieldByName('Average Invest').AsFloat * UAdj;
1449              // calculate Usage volume sensitive investment (U0002)
1450              USG.Line_Line_CCSVS:= dm_ICM.t_SwCost.FieldByName('VS Invest').AsFloat * UAdj;
1451            end;  // end [if t_swcost.findkey(U0002)]
1452
1453          if dm_ICM.t_SwCost.FindKey([SwitchClli, 'U0003']) then
1454            begin
1455              // calculate Usage investment (U0003)
1456              USG.Line_Trunk_Call  := dm_ICM.t_SwCost.FieldByName('Average Invest').AsFloat * UAdj;
1457              // calculate Usage volume sensitive investment (U0003)
1458              USG.Line_Trunk_CallVS:= dm_ICM.t_SwCost.FieldByName('VS Invest').AsFloat * UAdj;
1459            end;  // end [if t_swcost.findkey(U0003)]
1460
1461          if dm_ICM.t_SwCost.FindKey([SwitchClli, 'U0004']) then
1462            begin
1463              // calculate Usage investment (U0004)
1464              USG.Line_Trunk_CCS  := dm_ICM.t_SwCost.FieldByName('Average Invest').AsFloat * UAdj;
1465              // calculate Usage volume sensitive investment (U0004)
1466              USG.Line_Trunk_CCSVS:= dm_ICM.t_SwCost.FieldByName('VS Invest').AsFloat * UAdj;
1467            end;  // end [if t_swcost.findkey(U0004)]
1468
1469          if dm_ICM.t_SwCost.FindKey([SwitchClli, 'U0005']) then
1470            begin
1471              // calculate Usage investment (U0005)
1472              USG.Trunk_Line_Call  := dm_ICM.t_SwCost.FieldByName('Average Invest').AsFloat * UAdj;
1473              // calculate Usage volume sensitive investment (U0005)
1474              USG.Trunk_Line_CallVS:= dm_ICM.t_SwCost.FieldByName('VS Invest').AsFloat * UAdj;
1475            end;  // end [if t_swcost.findkey(U0005)]
1476
1477          if dm_ICM.t_SwCost.FindKey([SwitchClli, 'U0006']) then
1478            begin
```

```
1479              // calculate Usage investment (U0006)
1480              USG.Trunk_Line_CCS  := dm_ICM.t_SwCost.FieldByName('Average Invest').AsFloat * UAdj;
1481              // calculate Usage volume sensitive investment (U0006)
1482              USG.Trunk_Line_CCSVS:= dm_ICM.t_SwCost.FieldByName('VS Invest').AsFloat * UAdj;
1483            end;  // end [if t_swcost.findkey(U0006)]
1484
1485          if dm_ICM.t_SwCost.FindKey([SwitchClli, 'U0007']) then
1486            begin
1487              // calculate Usage investment (U0007)
1488              USG.Trunk_Trunk_Call  := dm_ICM.t_SwCost.FieldByName('Average Invest').AsFloat * UAdj;
1489              // calculate Usage volume sensitive investment (U0007)
1490              USG.Trunk_Trunk_CallVS:= dm_ICM.t_SwCost.FieldByName('VS Invest').AsFloat * UAdj;
1491            end;  // end [if t_swcost.findkey(U0007)]
1492
1493          if dm_ICM.t_SwCost.FindKey([SwitchClli, 'U0008']) then
1494            begin
1495              // calculate Usage investment (U0008)
1496              USG.Trunk_Trunk_CCS  := dm_ICM.t_SwCost.FieldByName('Average Invest').AsFloat * UAdj;
1497              // calculate Usage volume sensitive investment (U0008)
1498              USG.Trunk_Trunk_CCSVS:= dm_ICM.t_SwCost.FieldByName('VS Invest').AsFloat * UAdj;
1499            end;  // end [if t_swcost.findkey(U0008)]
1500
1501  { Per Bert Steele }
1502          if (Off_Type = 5) then // office type not tandem only
1503            begin
1504              if (hr = 'R') then // remote node
1505                begin  // combine investments for line/trunk usage features
1506                  USG.Line_Trunk_Call   := USG.Line_Trunk_Call + USG.Trunk_Trunk_Call;
1507                  USG.Line_Trunk_CCS    := USG.Line_Trunk_CCS  + USG.Trunk_Trunk_CCS;
1508                  USG.Trunk_Line_Call   := USG.Trunk_Line_Call + USG.Trunk_Trunk_Call;
1509                  USG.Trunk_Line_CCS    := USG.Trunk_Line_CCS  + USG.Trunk_Trunk_CCS;
1510
1511                  USG.Line_Trunk_CallVS := USG.Line_Trunk_CallVS + USG.Trunk_Trunk_CallVS;
1512                  USG.Line_Trunk_CCSVS  := USG.Line_Trunk_CCSVS  + USG.Trunk_Trunk_CCSVS;
1513                  USG.Trunk_Line_CallVS := USG.Trunk_Line_CallVS + USG.Trunk_Trunk_CallVS;
1514                  USG.Trunk_Line_CCSVS  := USG.Trunk_Line_CCSVS  + USG.Trunk_Trunk_CCSVS;
1515
1516                  // write remote lines record to 'actswtch' table
1517                  WriteActgRecord(t_SwAccount,CLLI,'000000','REMLINES','S',
```

```
1518                                                    Loops,Loops,Loops);
1519              end;  // end [if hr='R']
1520
1521            // write switch line to line call setup data to 'actswtch' table
1522            WriteActgRecord(t_SwAccount,CLLI,'221200','SWLLCS','S',
1523                            USG.Line_Line_Call,
1524                            USG.Line_Line_CallVS,
1525                            USG.Line_Line_Call - USG.Line_Line_CallVS);
1526            // write switch line to line minute of use data to 'actswtch' table
1527            WriteActgRecord(t_SwAccount,CLLI,'221200','SWLLMOU','S',
1528                            USG.Line_Line_CCS,
1529                            USG.Line_Line_CCSVS,
1530                            USG.Line_Line_CCS - USG.Line_Line_CCSVS) ;
1531            // write switch line to trunk call setup data to 'actswtch' table
1532            WriteActgRecord(t_SwAccount,CLLI,'221200','SWLTCS','S',
1533                            USG.Line_Trunk_Call,
1534                            USG.Line_Trunk_CallVS,
1535                            USG.Line_Trunk_Call - USG.Line_Trunk_CallVS);
1536          // write switch line to trunk minute of use data to 'actswtch' table
1537          WriteActgRecord(t_SwAccount,CLLI,'221200','SWLTMOU','S',
1538                            USG.Line_Trunk_CCS,
1539                            USG.Line_Trunk_CCSVS,
1540                            USG.Line_Trunk_CCS - USG.Line_Trunk_CCSVS);
1541          // write switch trunk to trunk call setup data to 'actswtch' table
1542          WriteActgRecord(t_SwAccount,CLLI,'221200','SWTLCS','S',
1543                            USG.Trunk_Line_Call,
1544                            USG.Trunk_Line_CallVS,
1545                            USG.Trunk_Line_Call  - USG.Trunk_Line_CallVS );
1546          // write switch trunk to line minute of use data to 'actswtch' table
1547          WriteActgRecord(t_SwAccount,CLLI,'221200','SWTLMOU','S',
1548                            USG.Trunk_Line_CCS ,
1549                            USG.Trunk_Line_CCSVS,
1550                            USG.Trunk_Line_CCS - USG.Trunk_Line_CCSVS);
1551          // write switch AMA data to 'actswtch' table
1552          WriteActgRecord(t_SwAccount,CLLI,'221200','SWAMA','S',
1553                            USG.AMA_Cost_Call,
1554                            USG.AMA_Cost_CallVS,
1555                            USG.AMA_Cost_Call - USG.AMA_Cost_CallVS);
1556
```

```
1557              if  (Off_Type = 5) and (hr = 'H') then // office type=end office only & Host node
1558                begin
1559                  // write switch trunk to trunk minute of use data to 'actswtch' table
1560                  WriteActgRecord(t_SwAccount,CLLI,'221200','SWTTMOU','S',
1561                                  USG.Trunk_Trunk_CCS,
1562                                  USG.Trunk_Trunk_CCSVS,
1563                                  USG.Trunk_Trunk_CCS - USG.Trunk_Trunk_CCSVS);
1564                  // write switch trunk to trunk call setup data to 'actswtch' table
1565                  WriteActgRecord(t_SwAccount,CLLI,'221200','SWTTCS','S',
1566                                  USG.Trunk_Trunk_Call ,
1567                                  USG.Trunk_Trunk_CallVS,
1568                                  USG.Trunk_Trunk_Call - USG.Trunk_Trunk_CallVS);
1569               end;  // end [if off_type=5 & hr='H']
1570            end  // end [if off_type=5]
1571          else if (Off_Type <> 5) then
1572            begin  // office type = 4 / 45(tandem only)
1573              // write switch tandem minute of use data to 'actswtch' table
1574              WriteActgRecord(t_SwAccount,CLLI,'221200','SWTDMOU','S',
1575                              USG.Trunk_Trunk_CCS,
1576                              USG.Trunk_Trunk_CCSVS,
1577                              USG.Trunk_Trunk_CCS - USG.Trunk_Trunk_CCSVS);
1578              // write switch tandem call setup data to 'actswtch' table
1579              WriteActgRecord(t_SwAccount,CLLI,'221200','SWTDCS','S',
1580                              USG.Trunk_Trunk_Call,
1581                              USG.Trunk_Trunk_CallVS,
1582                              USG.Trunk_Trunk_Call - USG.Trunk_Trunk_CallVS);
1583            end;  // end else office type = 4 /45 (tandem only)
1584        end;  // end [if OK]
1585
1586      dm_ICM.t_Nodes.next;  // get next records in switch nodes table (wire center)
1587
1588  end;  // end [while not t_nodes.eof]
1589
1590   dm_ICM.t_Switch.Close;     // close switch type table
1591   dm_ICM.t_Sw_EFI.Close;     // close EFI table
1592   dm_ICM.t_SwCost.Close;     // close switch investment table
1593   t_Features.Close;  // close features table
1594   f_OSPMaterial.Mat('CLOSE',0);   // close outside plant material table
1595   f_OSPLabor.Labor('CLOSE','XXXX');   // close outside plant labor table
```

```
1596
1597   {Create Host to Host table }
1598   t_Host := tTable.Create(Self);  // create table with name = t_host
1599    // get table name '3007' from associate module (from usetable table)
1600    TabName :=  f_associate.GetFileName(3007);
1601    With t_Host do
1602      begin
1603        if Active then Close;
1604        DatabaseName := ExtractFilePath(TabName);
1605        TableName := ExtractFilename(TabName);
1606        IndexFieldNames := '';
1607        IndexName := 'ByHubCLLI';
1608        TableType := ttDefault;
1609        Exclusive := True;
1610        Open;
1611      end;  // end [with t_host]
1612
1613    // set fields to search by in 'actswtch' table (t_swaccount)
1614    t_SwAccount.IndexFieldNames := 'CLLI;Account;Component';
1615    dm_ICM.t_Nodes.IndexFieldNames := 'CLLI';  // set field to search by in switch nodes (t_nodes) table
1616    dm_ICM.t_Nodes.First;  // go to first switch nodes (t_nodes) record
1617    while not dm_ICM.t_Nodes.Eof do  // execute until t_nodes end of file encountered
1618      begin
1619        Application.ProcessMessages;  // respond to user actions that have occurred
1620        CLLI       := dm_ICM.t_Nodes.FieldByName('CLLI').AsString;  // get switch nodes CLLI
1621        HCLLI      := dm_ICM.t_Nodes.FieldByName('Host CLLI').AsString;  // get switch nodes Host CLLI
1622        Off_Type   := dm_ICM.t_Nodes.FieldByName('Class').AsInteger;  // get switch nodes office type (class)
1623        if Off_Type = 5  // office type = end office only
1624         then begin
1625                WeAreWatching.LogEvent(4,'Process',' Update Tandem invest for - '+Clli);
1626                LCLLI      := CLLI;
1627                HCLLI      := dm_ICM.t_Nodes.FieldByName('Host CLLI').AsString;  // get switch nodes host CLLI
1628                WCBkmk := dm_ICM.t_Nodes.GetBookMark;  // set bookmark at current record
1629                while Off_Type = 5 do // office type = end office only
1630                  begin
1631                   if LCLLI <> HCLLI  // does previous CLLI = current host CLLI
1632                      then begin
1633                             if dm_ICM.t_Nodes.FindKey([HCLLI])  // find nodes host CLLI
1634                              then begin  // Host CLLI found
```

```
1635                                    Off_Type    := dm_ICM.t_Nodes.FieldByName('Class').AsInteger;  // get
1636  nodes office type
1637                                    LCLLI       := HCLLI;
1638                                    HCLLI       := dm_ICM.t_Nodes.FieldByName('Host CLLI').AsString;  // get
1639  nodes host CLLI
1640                                  end  // end [if t_nodes.findkey(HCLLI)]
1641                                else begin  // Host CLLI Not found
1642                                    HostFound := t_Host.FindKey([HCLLI]);  // retrieve host CLLI from Host
1643  to Host table
1644                                    T_Tandem.Findkey([t_Host.FieldByName('HubClli')]);
1645                                    if (not HostFound) or (HostFound and
1646                                     (T_Tandem.FieldByName('TndCompany').AsString <> 'O'))
1647                                     then begin
1648                                            // display error message
1649                                            MessageDlg(CLLI+'/'+ HCLLI + ' tandem does not exist.',
1650  mtInformation,
1651                                                       [mbOk],0);
1652                                            WeAreWatching.LogEvent(mError,'Process',CLLI+'/'+ HCLLI +
1653                                                         ' tandem does not exist.');
1654                                            Off_Type := 0;  // set office type to invalid value
1655                                          end  // end [if (not hostfound) or ...]
1656                                     else Off_Type := 0;
1657                                  end;  // end else host CLLI not found in nodes table
1658                             end  // end [if lclli<>hclli]
1659                          else begin  // else [lclli=hclli]
1660                            // display error message
1661                            MessageDlg(CLLI +'/'+ HCLLI +  ' doesn''t have a tandem conection.',
1662  mtInformation,
1663                                       [mbOk],0);
1664                            WeAreWatching.LogEvent(mNormal,'Process',CLLI +'/'+ HCLLI +  ' doesn''t have a
1665  tandem conection.');
1666                            Off_Type := 0;  // set office type to invalid value
1667                          end;  // end else [if lclli<>hclli]
1668                end;  // end [while off_type=5]
1669
1670              dm_ICM.t_Nodes.GotoBookmark(WCBkmk);  // return to bookmark
1671              dm_ICM.t_Nodes.FreeBookmark(WCBkmk);  // clear bookmark
1672
1673              if Off_type <> 0  // is office type valid
```

```
1674                then begin  // valid office types
1675                       {find tandem minute of use (SWTDMOU), host clli, and account
1676                        record.in 'actswtch' table}
1677                       if t_SwAccount.FindKey([HCLLI, '221200','SWTDMOU'])
1678                        then begin
1679                               // load investment to usage
1680                               USG.Trunk_Trunk_CCS   := t_SwAccount.FieldByName('Total Inv').AsFloat;
1681                               // load volume sensitive investment to usage
1682                               USG.Trunk_Trunk_CCSVS := t_SwAccount.FieldByName('VS Inv').AsFloat;
1683                             end  // end [if t_swaccount.findkey (SWTDMOU)]
1684                        else begin  // did not find SWTDMOU, host clli and account record
1685                               // display error message
1686                               MessageDlg(HCLLI + ' didn''t have SWTDMOU', mtInformation, [mbOk],0);
1687                               USG.Trunk_Trunk_CCS   := 0.0;  // set investment usage to zero
1688                               USG.Trunk_Trunk_CCSVS := 0.0;  // set volume sensitive investment usage to
1689 zero
1690                             end;  // end else (did not find SWTDMOU...)
1691                       {find tandem call setup (SWTDCS), host clli, and account
1692                        record in 'actswtch' table}
1693                       if t_SwAccount.FindKey([HCLLI, '221200','SWTDCS'])
1694                        then begin
1695                               // load investment to usage
1696                               USG.Trunk_Trunk_Call   := t_SwAccount.FieldByName('Total Inv').AsFloat;
1697                               // load volume sensitive investment to usage
1698                               USG.Trunk_Trunk_CallVS := t_SwAccount.FieldByName('VS Inv').AsFloat;
1699                             end  // end [if t_swaccount.findkey (SWTDCS)]
1700                        else begin  // did not find SWTDCS, host clli and account record
1701                               // display error message
1702                               MessageDlg(HCLLI + ' didn''t have SWTDCS', mtInformation, [mbOk],0);
1703                               USG.Trunk_Trunk_Call   := 0.0;  // set investment usage to zero
1704                               USG.Trunk_Trunk_CallVS := 0.0;  // set volume sensitive investment usage to
1705 zero
1706                             end;  // end else (did not find SWTDCS...)
1707
1708                       // write tandem call setup data to 'actswtch' table
1709                       WriteActgRecord(t_SwAccount,CLLI,'221200','SWTDCS','S',
1710                                       USG.Trunk_Trunk_Call,
1711                                       USG.Trunk_Trunk_CallVS,
1712                                       USG.Trunk_Trunk_Call - USG.Trunk_Trunk_CallVS);
```

```
1713                        // write tandem minute of use data to 'actswtch' table
1714                        WriteActgRecord(t_SwAccount,CLLI,'221200','SWTDMOU','S',
1715                                        USG.Trunk_Trunk_CCS,
1716                                        USG.Trunk_Trunk_CCSVS,
1717                                        USG.Trunk_Trunk_CCS - USG.Trunk_Trunk_CCSVS);
1718                      end;  // end [if off_type<>0]
1719              end;  // end [if off_type=5]
1720
1721        dm_ICM.t_Nodes.Next;  // get next record in switch nodes table (t_nodes)
1722      end;  // end [while not t_nodes.eof]
1723
1724    t_Tandem.Close;  //close tandems table
1725    t_SwAccount.Close;  // close 'actswtch' table
1726    dm_ICM.t_Nodes.Close;      // close switching nodes table
1727    t_host.Close;       // close host to host table
1728    t_host.Free;        // free up memory used for t_host table
1729
1730    f_associate.SetFileName(101, 'NO'); {set Switch process to no (or done)}
1731    f_associate.SetFileName(102, 'YES');{set SS7 process to yes}
1732    Hide;  // make the switch form invisible
1733  end;  // end [tf_switching.process procedure]
1734
1735  procedure Tf_Switching.Create_ACC_Switch_Table;
1736   begin
1737  { Create an empty table to hold the switching investment}
1738
1739    with t_SwAccount do  // set table variable name and set up table options
1740      begin
1741        Active := False;
1742        DatabaseName := 'modata';
1743        TableName := 'actswtch';
1744        IndexFieldNames := '';
1745        IndexName := '';
1746        TableType := ttParadox;
1747        Exclusive := False;
1748        with FieldDefs do
1749          begin  // define fields within table
1750            Clear;
1751            Add ('CLLI',         FTString, 11, True);
```

```
1752            Add ('Account',      FTString,  6, True);
1753            Add ('Component',    FTString, 16, True);
1754            Add ('Category',     FTString,  1, False);
1755            Add ('Total Inv',    FTFloat,   0, False);
1756            Add ('VS Inv',       FTFloat,   0, False);
1757            Add ('VIS Inv',      FTFloat,   0, False);
1758          end;  // end [with fielddefs]
1759
1760        with IndexDefs do
1761          (define index of table (used in 'key' searching - what fields to use
1762           when a search is requested))
1763          begin
1764            Clear;
1765            Add('Main', 'CLLI;Account;Component', [ixPrimary, ixUnique]);
1766            Add('ByAcc', 'Account;Component', [ixCaseInsensitive]);
1767          end;  // end [with indexdefs]
1768
1769        CreateTable;  // create the defined table
1770        Open;  // open the defined table
1771      end;   // end [with t_swacount]
1772   end;  // end procedure tf_switching.create_acc_switch_table
1773
1774  procedure Tf_Switching.FormResize(Sender: TObject);
1775  // Resizes the form per user input
1776  begin
1777  if ClientWidth < 350
1778      then ClientWidth := 350;
1779
1780   dbg_Switch.Width := ClientWidth-20;
1781   ts_Switch.Width := ClientWidth;
1782   dbn_Switch.Width := 191;
1783
1784   dbg_Switch.Left := 10;
1785   ts_Switch.Left  := 0;
1786   if ClientWidth > (dbn_Switch.Width + cb_Assoc.Width + 5)
1787     then dbn_Switch.Left := (ClientWidth - (dbn_Switch.Width + cb_Assoc.Width + 5)) div 2
1788     else dbn_Switch.Left := 1;
1789
1790   cb_Assoc.Left    := dbn_Switch.Left + dbn_Switch.Width + 5;
```

```
1791    p_Progress.Left := (ClientWidth - p_Progress.Width) div 2;
1792
1793    dbg_Switch.Top := l_CopyRight.Height+2;
1794    ts_Switch.Top  := ClientHeight - ts_Switch.Height;
1795    dbn_Switch.Top := ClientHeight - (ts_Switch.Height + dbn_Switch.Height + 5);
1796    cb_Assoc.Top   :=  dbn_Switch.Top + (dbn_Switch.Height-cb_Assoc.Height) div 2;
1797    p_Progress.Top := l_Copyright.Height + 5;
1798
1799    dbg_Switch.Height :=  dbn_Switch.Top - (dbg_Switch.Top+5);
1800  end;  // end procedure tf_switching.formresize
1801
1802  procedure Tf_Switching.Exit1Click(Sender: TObject);
1803  // This procedure is accessed when the user selects EXIT from the Main Menu
1804  begin
1805    if RunFlag then  f_associate.SetFileName(101, 'YES'); {set Switch process to yes}
1806    Update_cb_Investment := False;
1807    Hide;  // make switch form invisible
1808  end;  // end tf_switching.exit1click procedure
1809
1810  procedure Tf_Switching.SetTableFilter(Tab : Integer);
1811  {This procedure sets up and displays related table information per user selection.
1812  It is called from procedures :  ts_switchchange, open1click, new1click,
1813  import1click, and print1click.  'Tab' is the index # of selected tab from the form. }
1814  begin
1815    dbg_Switch.Options:= [dgEditing,dgTitles,dgIndicator,dgColumnResize,dgColLines,
1816                          dgRowLines,dgTabs,dgConfirmDelete,dgCancelOnExit];
1817     case Tab of
1818      0 : begin  // Switch tab selected
1819             // set the caption (form title) to include file name of related tab
1820             Caption := 'Switch ['+ ExtractFilename(TabName)+']';
1821             dbg_Switch.Height :=  dbn_Switch.Top - (dbg_Switch.Top+5);
1822          end;  // end case tab=0 (Switch)
1823      1 : begin // EFI tab selected
1824             // set the caption (form title) to include file name of related tab
1825             Caption := 'EFI ['+ ExtractFilename(TabName)+']';
1826             // set options for table data display
1827             dbg_Switch.Height :=  dbn_Switch.Top - (dbg_Switch.Top+5);
1828          end;  // end case tab=1 (EFI)
1829      2 : begin  // Termination tab selected
```

```
1830            // set the caption (form title) to include file name of related tab
1831            Caption := 'Termination ['+ ExtractFilename(TabName)+']';
1832            dbg_Switch.Height :=  dbn_Switch.Top - (dbg_Switch.Top+5);
1833
1834            dm_ICM.t_Switch.IndexFieldNames := 'Category';  // set to sort on 'category' field in t_switch
1835    table
1836            dm_ICM.t_Switch.CancelRange;  // cancel t_switch table search ranges
1837            dm_ICM.t_Switch.SetRange(['T'], ['T']);  // set filters to 'termination' search
1838            dm_ICM.t_Switch.Refresh;  // get switch data & update view on screen
1839          end;  // end case tab=2 (Termination)
1840      3 : begin  // Usage tab selected
1841            // set the caption (form title) to include file name of related tab
1842            Caption := 'Usage ['+ ExtractFilename(TabName)+']';
1843            dbg_Switch.Height :=  dbn_Switch.Top - (dbg_Switch.Top+5);
1844
1845            dm_ICM.t_Switch.IndexFieldNames := 'Category';  // set to sort on 'category' field in t_switch
1846    table
1847            dm_ICM.t_Switch.CancelRange;  // cancel t_switch table search ranges
1848            dm_ICM.t_Switch.SetRange(['U'], ['U']);  // set filters for 'usage' search
1849            dm_ICM.t_Switch.Refresh;  // get switch data & update view on screen
1850          end;  // end case tab=3 (Usage)
1851      4 : begin  // Features tab selected
1852            // set the caption (form title) to include file name of related tab
1853            Caption := 'Features ['+ ExtractFilename(TabName)+']';
1854            dbg_Switch.Height :=  dbn_Switch.Top - (dbg_Switch.Top+5);
1855
1856            dm_ICM.t_Switch.IndexFieldNames := 'Category';  // set to sort on 'category' field in t_switch
1857    table
1858            dm_ICM.t_Switch.CancelRange;  // cancel t_switch table search ranges
1859            dm_ICM.t_Switch.SetRange(['F'], ['F']);  // set filters for 'feature' search
1860            dm_ICM.t_Switch.Refresh;  // get switch data & update view on screen
1861          end;  // end case tab=4 (features)
1862      5 : begin  // Investment Tab selected
1863            // set the caption (form title) to include file name of related tab
1864            Caption := 'Switch Investment ['+ ExtractFilename(TabName)+']';
1865            dbg_Switch.Height :=  dbn_Switch.Top - (dbg_Switch.Top+5);
1866
1867            dm_ICM.t_Switch.IndexFieldNames := 'Category';  // set to sort on 'category' field in t_switch
1868    table
```

```
1869            dm_ICM.t_Switch.CancelRange;  // cancel t_switch table search ranges
1870            dm_ICM.t_Switch.SetRange(['L'], ['L']);  // set filters for 'line' search
1871            dm_ICM.t_Switch.Refresh;  // get switch data & update view on screen
1872          end;  // end case tab=5 (Investment)
1873      6 : begin  // Miscellaneous tab selected
1874            // set the caption (form title) to include file name of related tab
1875            Caption := 'Miscellaneous ['+ ExtractFilename(TabName)+']';
1876            dbg_Switch.Height :=  dbn_Switch.Top - (dbg_Switch.Top+5);
1877
1878            dm_ICM.t_Switch.IndexFieldNames := 'Category';  // set to sort on 'category' field in t_switch
1879  table
1880            dm_ICM.t_Switch.CancelRange;  // cancel t_switch table search ranges
1881            dm_ICM.t_Switch.SetRange(['M'], ['M']); // set filters for 'miscellaneous' search
1882            dm_ICM.t_Switch.Refresh;  // get switch data & update view on screen
1883          end;  // end case tab=6 (miscellaneous)
1884      7 : begin  //RFPC0060 RTU Fees tab selected
1885            // set the caption (form title) to include file name of related tab
1886            Caption := 'RTU Fees ['+ ExtractFilename(TabName)+']';
1887            dbg_Switch.Height :=  dbn_Switch.Top - (dbg_Switch.Top+5);
1888
1889            dm_ICM.t_Switch.IndexFieldNames := 'Category';  // set to sort on 'category' field in t_switch
1890  table
1891            dm_ICM.t_Switch.CancelRange;  // cancel t_switch table search ranges
1892            dm_ICM.t_Switch.SetRange(['R'], ['R']); // set filters for 'miscellaneous' search
1893            dm_ICM.t_Switch.Refresh;  // get switch data & update view on screen
1894          end;  // end case tab=7 (RTU)
1895     end;  // end case Tab
1896  end;  // end procedure tf_switching.settablefilter
1897
1898  procedure Tf_Switching.ts_SwitchChange(Sender: TObject; NewTab: Integer;
1899  {This procedure is executed when a new Tab is selected from the form.  'NewTab'
1900   is the index of the tab selected. }
1901    var AllowChange: Boolean);
1902  begin
1903     Update_cb_Investment := False;  // don't update investment type box with new data
1904     dm_ICM.t_Switch.IndexFieldNames := '';  // clear table index
1905     dm_ICM.t_Switch.IndexName := '';  // clear table index
1906
1907     if dm_ICM.t_Switch.Active  // is t_switch table open
```

```
1908      then begin  // yes, t_switch is open
1909             dm_ICM.t_Switch.CancelRange;  // cancel t_switch search ranges
1910             dm_ICM.t_Switch.Close;  // close t_switch table
1911           end;  // end [if t_switch.active]
1912
1913     if cb_Assoc.Checked  // is 'update association' box checked
1914       then begin  // yes, it's checked
1915              case ts_Switch.TabIndex of  // which Tab in switch form is selected
1916                0 : f_associate.SetFileName(2001, TabName);  // set filename for switch (2001)
1917                1 : f_associate.SetFileName(2011, TabName);  // set filename for EFI (2011)
1918                {set filename for switch investment (Termination, Usage,
1919                 Feature, Investment/Line, Miscellaneous) (3003)}
1920              2..7 :   if Wholesale
1921                         then f_associate.SetFileName(3003, TabName)   // Wholesale Switch Investment table
1922  filename and location using index 3003
1923                         else f_associate.SetFileName(4003, TabName);  // Retail  Switch Investment table
1924  filename and location using index 4003
1925
1926              end;  // end [case ts_switch.tabindex]
1927              cb_Assoc.Checked := False;  // clear update association check box
1928            end;  // end [if cb_assoc.checked]
1929
1930     case NewTab of  // set new form defaults per tab selected
1931       0 : begin  // switch tab selected
1932              TabName :=  f_associate.GetFileName(2001);  // get filename for switch table (2001)
1933              Tab1.Caption := ts_Switch.Tabs[Newtab];
1934           end;  // end case newtab=0 (switch)
1935       1 : begin  // EFI tab selected
1936              TabName :=  f_associate.GetFileName(2011);  // get filename for EFI table (2011)
1937              Tab1.Caption := ts_Switch.Tabs[Newtab];
1938           end;  // end case newtab=1 (EFI)
1939     2..7 : begin  // Termination, Usage, Features, Investment/Line, Miscellaneous tab selected
1940              if Wholesale
1941                then TabName :=  f_associate.GetFileName(3003)   // Wholesale Switch Investment table
1942  filename and location using index 3003
1943                else TabName :=  f_associate.GetFileName(4003);  // Retail  Switch Investment table filename
1944  and location using index 4003
1945              Tab1.Caption := 'Switch Investment';
1946           end;  // end case newtab=2..6
```

```
1947      end;  // end [case newtab]
1948
1949      if TabName = ''  // no tab selected
1950        then begin
1951               exit;  // exit from switchchange procedure
1952             end;  // end [if tabname]
1953
1954      Caption := Caption + ExtractFilename(TabName)+']';  // set form title using related filename
1955        With dm_ICM.t_Switch do  // open t_switch table (switch, EFI, or switch investment)
1956         begin
1957           DatabaseName := ExtractFilePath(TabName);
1958           TableName := ExtractFilename(TabName);
1959           TableType := ttDefault;
1960           Exclusive := True;
1961           Open;
1962         end;  // end [with t_switch]
1963
1964      {If not switch or EFI tab then set flag to update investment type box with new data}
1965      if NewTab > 1 then Update_cb_Investment := True;
1966      SetTableFilter(NewTab);  // set up & display related newtab table data
1967  end;  // end procedure tf_switching.ts_switchchange
1968
1969  procedure Tf_Switching.FormShow(Sender: TObject);
1970  {This procedure is executed prior to switch form being displayed.}
1971  begin
1972    RunFlag := False;  // set flag to not process switch data
1973
1974    l_Copyright.Caption := f_associate.GetFileName(5);   // get copyright data
1975
1976    {Features List table }
1977    TabName :=  f_associate.GetFileName(2007);  // get features table (2007) filename and location
1978    With t_Features do
1979      begin
1980        if Active then Close;
1981        DatabaseName := ExtractFilePath(TabName);
1982        TableName := ExtractFilename(TabName);
1983        TableType := ttDefault;
1984        Exclusive := False;
1985      end;  // end [with t_features]
```

```
1986
1987    case Mode of  // set form information depending on entering or exiting form
1988     1 : begin  // entering form
1989           Width  := 430;
1990           Height := 410;
1991           p_Progress.Visible := False;     // hide progress bar
1992           cb_Assoc.Visible   := True;      // show update association check box
1993           dbg_Switch.Visible := True;      // show table display area
1994           ts_Switch.Visible  := True;      // show tab selection area
1995           dbn_Switch.Visible := True;      // show navigator buttons
1996           Update_cb_Investment := False;  // do not allow update of investment type box
1997           cb_Assoc.Checked := False;       // default update association check box to not set
1998           dm_ICM.t_Switch.IndexFieldNames := ''; // clear switch table index
1999           dm_ICM.t_Switch.IndexName := '';       // clear switch table index
2000           ts_Switch.TabIndex := 0;         // set tabs to switch
2001           Tab1.Caption := ts_Switch.Tabs[0]; // Set help menu caption
2002           TabName :=  f_associate.GetFileName(2001);  // get switch table (2001) filename and location
2003           With dm_ICM.t_Switch do  // open switch table
2004             begin
2005               if Active then Close;
2006               DatabaseName := ExtractFilePath(TabName);
2007               TableName := ExtractFilename(TabName);
2008               TableType := ttDefault;
2009               Exclusive := True;
2010               Open;
2011             end; // end [with t_switch]
2012
2013
2014           Caption := 'Switch [' + ExtractFilename(TabName)+']';  // set form title per tab selection
2015           dm_ICM.t_Switch.First;  // goto first record in switch/switch investment table
2016        end;  // end [case mode=1]
2017      2: begin  // exiting form
2018           Width := 300;
2019           Height:= 150;
2020           p_Progress.Visible := True;   // show progress bar
2021           cb_Assoc.Visible   := False;  // hide update association check box
2022           dbg_Switch.Visible := False;  // hide table display area
2023           ts_Switch.Visible  := False;  // hide tab selection area
2024           dbn_Switch.Visible := False;  // hide navigator buttons
```

```
2025          end;  // end [case mode=2]
2026      end;  // end [case mode]
2027    end;  // end procedure tf_switching.formshow
2028
2029    procedure Tf_Switching.FormHide(Sender: TObject);
2030    begin
2031      if dm_ICM.t_Switch.Active then dm_ICM.t_Switch.Close;
2032      if cb_Assoc.Checked
2033        then case ts_Switch.TabIndex of
2034               0 : f_associate.SetFileName(2001, TabName);
2035               1 : f_associate.SetFileName(2011, TabName);
2036            2..7 : f_associate.SetFileName(3003, TabName);
2037             end;
2038       cb_Assoc.Checked := False;
2039    end;  // end procedure Tf_Switching.FormHide
2040
2041    procedure Tf_Switching.Open1Click(Sender: TObject);
2042    begin
2043      od_Switch.Filter := 'Database|*.db;*.dbf';
2044      if Not od_Switch.execute then exit;
2045      TabName := od_Switch.FileName;
2046
2047      with dm_ICM.t_Switch do
2048        begin
2049          Active := False;
2050          DatabaseName := ExtractFilePath(TabName);
2051          TableName := ExtractFilename(TabName);
2052          TableType := ttDefault;
2053          Exclusive := True;
2054          Open;
2055        end;  // end [with t_switch]
2056      SetTableFilter(ts_Switch.TabIndex);
2057      WeAreWatching.LogEvent(2,'Open1Click','Alternate table opened for Switch');
2058    end;  // end procedure Tf_Switching.Open1Click
2059
2060    procedure Tf_Switching.New1Click(Sender: TObject);
2061    begin
2062      sd_Switch.Filter := 'Database|*.db';
2063      if Not sd_Switch.execute then exit;
```

```
2064    TabName := sd_Switch.FileName;
2065    cb_Assoc.Checked := True;
2066
2067    case ts_Switch.TabIndex of
2068     0 : begin
2069              with dm_ICM.t_Switch do
2070                begin
2071                  Active := False;
2072                  DatabaseName := ExtractFilePath(TabName);
2073                  TableName := ExtractFilename(TabName);
2074                  TableType := ttParadox;
2075                  Exclusive := True;
2076                  with FieldDefs do
2077                    begin
2078                      Clear;
2079                      Add ('Switch Type',  FTString, 12, True);
2080                      Add ('ID',           FTInteger, 0, False);
2081                      Add ('Host/Rem',     FTString, 1,  True);
2082                    end;  // end [with fielddefs]
2083
2084                  with IndexDefs do
2085                     begin
2086                       Clear;
2087                       Add('Main', 'Switch Type', [ixPrimary, ixUnique]);
2088                     end;  // end [with indexdefs]
2089
2090                  CreateTable;
2091                  Open;
2092                end;  // end [with t_switch]
2093        end;  // end [case =0]
2094     1 : begin
2095              with dm_ICM.t_Switch do
2096                begin
2097                  Active := False;
2098                  DatabaseName := ExtractFilePath(TabName);
2099                  TableName := ExtractFilename(TabName);
2100                  TableType := ttParadox;
2101                  Exclusive := True;
2102                  with FieldDefs do
```

```
2103                    begin
2104                      Clear;
2105                      Add ('Switch Type',  FTString, 12,  True);
2106                      Add ('Size',         FTFloat,   0,  True);
2107                      Add ('EFI-I',        FTFloat,   0,  False);
2108                      Add ('EFI-EF',       FTFloat,   0,  False);
2109                      Add ('IAF',          FTFloat,   0,  False);
2110                      Add ('Power Inv',    FTFloat,   0,  False);
2111                      Add ('Test Inv',     FTFloat,   0,  False);
2112                      Add ('Loop test',    FTFloat,   0,  False);
2113                    end;  // end [with fielddefs]
2114
2115                  with IndexDefs do
2116                     begin
2117                       Clear;
2118                       Add('Main', 'Switch Type;Size', [ixPrimary, ixUnique]);
2119                     end;  // end [with indexdefs]
2120
2121                  CreateTable;
2122                  Open;
2123                end;  // end [with t_switch]
2124          end;  // end [case=1]
2125   2..7 : begin
2126                with dm_ICM.t_Switch do
2127                  begin
2128                    Active := False;
2129                    DatabaseName := ExtractFilePath(TabName);
2130                    TableName := ExtractFilename(TabName);
2131                    TableType := ttParadox;
2132                    Exclusive := True;
2133                    with FieldDefs do
2134                      begin
2135                        Clear;
2136                        Add ('Clli',             FTString, 11, True);
2137                        Add ('Investment Type',  FTString, 6,  True);
2138                        Add ('Category',         FTString, 1,  True);
2139                        Add ('Average Invest',   FTFloat,  0,  False);
2140                        Add ('VS Invest',        FTFloat,  0,  False);
2141                      end;  // end [with fielddefs]
```

```
2142
2143                  with IndexDefs do
2144                     begin
2145                       Clear;
2146                       Add('Main', 'Clli;Investment Type', [ixPrimary, ixUnique,ixCaseInsensitive]);
2147                       Add('Cat', 'Category', [ixCaseInsensitive]);
2148                     end;  // end [indexdefs]
2149
2150                  CreateTable;
2151                  Open;
2152                end;  // end [with t_switch]
2153          end;  // end [case=2..7]
2154    end;  // end [case ts_Switch.TabIndex]
2155
2156    SetTableFilter(ts_Switch.TabIndex);
2157  end;  // end procedure Tf_Switching.New1Click
2158
2159  procedure Tf_Switching.Import1Click(Sender: TObject);
2160  var Success: Boolean;
2161  begin
2162   dm_ICM.t_Switch.Close;
2163   ImportDataCall('', AliasPath+'\'+dm_ICM.t_Switch.Tablename, 10);
2164   dm_ICM.t_Switch.Open;
2165
2166   SetTableFilter(ts_Switch.TabIndex);
2167   WeAreWatching.LogEvent(2,'Import1Click','Imported into Switch table.');
2168  end;  // end procedure Tf_Switching.Import1Click
2169
2170  procedure Tf_Switching.Export1Click(Sender: TObject);
2171  begin
2172    ExportTable( dm_ICM.t_Switch);
2173  end;  // end procedure Tf_Switching.Export1Click
2174
2175  procedure Tf_Switching.PrintSwitch( Open : Boolean);
2176   begin
2177     if Open
2178       then begin
2179              TabName :=  f_associate.GetFileName(2001);
2180              With dm_ICM.t_Switch do
```

```
2181                begin
2182                  DatabaseName := ExtractFilePath(TabName);
2183                  TableName := ExtractFilename(TabName);
2184                  TableType := ttParadox;
2185                  IndexName := '';
2186                  IndexFieldNames := '';
2187                  Exclusive := True;
2188                  Open;
2189                end;  // end [with t_switch]
2190              end;  // end [if open]
2191
2192     GenericPrintTable(dm_ICM.t_Switch, Nil, 0, False,False, 'Switches');
2193     if Open then dm_ICM.t_Switch.Close;
2194   end;  // end procedure Tf_Switching.PrintSwitch
2195
2196  procedure Tf_Switching.PrintSwEFI( Open : Boolean);
2197   begin
2198     if Open
2199       then begin
2200              TabName :=  f_associate.GetFileName(2011);
2201              With dm_ICM.t_Switch do
2202               begin
2203                 DatabaseName := ExtractFilePath(TabName);
2204                 TableName := ExtractFilename(TabName);
2205                 TableType := ttParadox;
2206                 IndexName := '';
2207                 IndexFieldNames := '';
2208                 Exclusive := True;
2209                 Open;
2210               end;  // end [with t_switch]
2211             end;  // end [if open]
2212
2213     GenericPrintTable(dm_ICM.t_Switch, Nil, 0, False,False, 'Switch EF&I');
2214     if Open then dm_ICM.t_Switch.Close;
2215   end;  // end procedure Tf_Switching.PrintSwEFI
2216
2217  procedure Tf_Switching.PrintSwCost( Open : Boolean);
2218   begin
2219     if Open
```

```
2220        then begin
2221               if Wholesale
2222                 then TabName :=  f_associate.GetFileName(3003)   // Wholesale Switch Investment table
2223  filename and location using index 3003
2224                 else TabName :=  f_associate.GetFileName(4003);  // Retail  Switch Investment table filename
2225  and location using index 4003
2226              With dm_ICM.t_Switch do
2227               begin
2228                 DatabaseName := ExtractFilePath(TabName);
2229                 TableName := ExtractFilename(TabName);
2230                 TableType := ttParadox;
2231                 Exclusive := True;
2232                 Open;
2233               end;  // end [with t_switch]
2234            end;  // end [if open]
2235
2236     GenericPrintTable(dm_ICM.t_Switch, Nil, 0, False,False, 'Switch Investments');
2237     if Open then dm_ICM.t_Switch.Close;
2238   end;  // end procedure Tf_Switching.PrintSwCost
2239
2240  procedure Tf_Switching.Print1Click(Sender: TObject);
2241  begin
2242    case ts_Switch.TabIndex of
2243     0 : GenericPrintTable(dm_ICM.t_Switch, Nil, 0, False,True, 'Switches');
2244     1 : GenericPrintTable(dm_ICM.t_Switch, Nil, 0, False,True, 'Switch EF&I');
2245  2..6 : begin
2246           dm_ICM.t_Switch.CancelRange;
2247           GenericPrintTable(dm_ICM.t_Switch, Nil, 0, False,True, 'Switch Investment');
2248           dm_ICM.t_Switch.IndexFieldNames :='Category';
2249           SetTableFilter(ts_Switch.TabIndex);
2250         end;  // end [case 2..6]
2251    end;  // end [case ts_switch.tabindex]
2252  end;  // end procedure Tf_Switching.Print1Click
2253
2254  procedure Tf_Switching.FormActivate(Sender: TObject);
2255  begin
2256    UnitName := 'Switch';
2257  end;  // end procedure Tf_Switching.FormActivate
2258
```

```
2259  procedure Tf_Switching.Tab1Click(Sender: TObject); // Help file connection
2260  begin
2261    Case ts_Switch.TabIndex of
2262      0 : GoHelp ('Datatables.hlp', 406);
2263      1 : GoHelp ('Datatables.hlp', 407);
2264      2 : GoHelp ('Datatables.hlp', 410);
2265      3 : GoHelp ('Datatables.hlp', 410);
2266      4 : GoHelp ('Datatables.hlp', 410);
2267      5 : GoHelp ('Datatables.hlp', 410);
2268      6 : GoHelp ('Datatables.hlp', 410);
2269      7 : GoHelp ('Datatables.hlp', 410);
2270    end;
2271
2272  end;
2273
2274  procedure Tf_Switching.Common1Click(Sender: TObject);
2275  begin
2276    GoHelp ('MenuIt.hlp', 324);
2277  end;
2278
2279  end.
2280
```