Infor Omni-Channel Campaign Management Data Mart Implementation Guide

Publication information

Release: 10.1

Publication Date: September 21, 2015

Contents

Introduction

1

This *Infor Campaign Management Data Mart Implementation Guide* provides in-depth technical information on how to configure and populate the data mart used by all Infor Campaign Management applications, including Infor Enterprise Insight applications.

Infor Campaign Management

Infor Campaign Management, a component of Customer Relationship Management (CRM), serves as an integrated database campaign management solution for planning, executing, and monitoring permission-based marketing campaigns across multiple touch points.

Infor Campaign Management provides the best-of-breed campaign management, sophisticated database management, integrated OLAP and predictive analytics, the ability to execute campaigns across multiple touch points, and built-in closed-loop reporting. The solution's automated global business rules and system-wide opt-in and opt-out functionality helps marketers enforce company marketing policies and regulatory compliance. Its tightly integrated permission-based campaign management capabilities, robust analytics, and intuitive interface help them rapidly develop highly optimized campaigns. Closed-loop analysis and predictive behavior patterns make it easy to understand campaign responses, improve targeting, and drive intelligence into subsequent campaigns.

Infor Campaign Management Architecture

Infor Campaign Management is based on a modern architecture that provides scalability to billions of records to handle large enterprise requirements. It delivers the high performance necessary to scale up to thousands of users while delivering fast, interactive response times.

The Infor Campaign Management multi-tier architecture employs an ultra-thin web browser client, a middle-tier application server implemented in Java, and an underlying industry-standard relational database server. This enables it to integrate customer data from a wide variety of sources and provide a comprehensive view of customers and their behavior.

These sources include:

- Touchpoint systems
- ERP/operational applications
- E-commerce web logs
- Legacy systems
- Data warehouses
- Data marts
- Third-party data sources

Major Themes

The keystones of Infor Campaign Management architecture include the following major themes:

- Integrated and comprehensive technology stack. Infor Campaign Management applications are built on an integrated platform that provides an end-to-end technology stack. This stack provides the following capabilities:
  - Data integration
  - Data management
  - Advanced analysis (including OLAP, data mining and list analytics)
  - Campaign planning and design
  - Campaign execution
- The entire platform is seamlessly integrated through rich, common metadata, which enables unparalleled productivity and simplicity in implementation as well as use.
- Easy to use, web-based architecture.
- The Infor Campaign Management thin web browser client is easy for business users to learn, easy for implementers to deploy, and scalable to large numbers of users.
- Designed for rapid and incremental deployment.
- Infor Campaign Management is explicitly designed to enable rapid initial implementation and to quickly respond to changing business needs. Adaptive growth is a direct consequence of the investment in rich, common metadata that spans the system from data integration, to data mart construction and management, and to the design of end-user interfaces.
- Structured for high performance and enterprise scalability.
- Infor Campaign Management achieves high performance and massive scalability without the use of any proprietary or esoteric technologies. (These technologies are typically hard to manage and often result in increased cost of ownership.) Instead, it runs entirely on industry standard technologies:
  - Relational database management systems (RDBMS) from IBM, Microsoft and Oracle
  - Java
  - Microsoft and Netscape browsers
  - ODBC and JDBC data access technologies

Other Features

Infor Campaign Management includes a scalable data mart, an integrated metadata repository and an extensible data extraction infrastructure. It includes powerful data transformation technologies, a scalable application server, high-performance analytic components that deliver ROLAP, data mining and list analysis capabilities, and a complete campaign management system. It also includes an execution engine for Infor Email Marketing, and an engine for Infor Interaction Advisor (Real-Time).

The only external software components required are: a relational database engine, a web server, and a web browser.

Localized versions of Infor Campaign Management software support data mart applications in which the entire web-based interface appears in a supported local language. A localized data mart can contain data values in any one of the currently supported languages, locales, and code sets.

Infor Campaign Management Documentation

The Infor Campaign Management product documentation includes four manuals and two online help systems. Each of these manuals is updated whenever needed. Most are updated for each maintenance release, but occasionally one or two will not need updating, and thus bear an earlier version number. If you are not sure whether or not you have all the correct documentation for the release your company purchased, contact Infor Customer Support.

Installation Guide

The *Infor Campaign Management Installation Guide* is intended for database administrators who install, configure, and maintain the Infor Campaign Management. The guide is designed to be used in conjunction with our product release notes, plus the appropriate installation and configuration manuals for your existing hardware and software. The following list displays the names of software and related manuals, that can be used with Infor Campaign Management.

DB2 on AIX	Quick Beginnings: DB2 for UNIX and DB2 Administration Guide, Volumes 1 through 3. Or, the Installation Guide and the Performance Management Guide for AIX.
Oracle	The appropriate installation guide for Oracle on your operating system. Or, the installation guide for your operating system if that operating system is not pre installed. The instructions for configuring kernel resources such as Net Configuration Assistant, or Oracle11g Reference.

SQL Server	SQL Server Administration Guide, or the appropriate administration guide for your Windows version.
All platforms	The installation and configuration instructions for your RAID disk-management equipment and software.

Note: Please follow the configuration recommendations that are suggested in this guide. If specific instructions do not appear for a particular configuration step or option, default values are acceptable.

Quick Implementation Guide

The *Infor Campaign Management Quick Implementation Guide* is designed to help beginning implementers get started with Infor Campaign Management by using and implementing the sample data mart shipped with the product. It includes the following:

- An overview of the product architecture
- Information on data mart design and creation
- An explanation of how extraction works
- Step-by-step procedures for creating Rows and Columns web pages, reporting and analysis, lists, and campaigns

Topic Implementation Guide

Formerly titled Configuring Applications, the *Infor Campaign Management Topic Implementation Guide* is intended for implementers and Professional Services personnel charged with creating applications that run on an Infor Campaign Management EpiCenter. The manual overviews Infor Campaign Management functionality, architecture, and administration, and provides in-depth technical information on how to configure the Infor Campaign Management topics required for Campaign Management and analysis.

Viewing Release Notes and Manuals Online

You can view our product documentation and Release Notes on any machine that has Acrobat Reader running.

When you install documentation on Windows, the installation wizard adds shortcuts to the Windows **Start** menu for easy viewing access. Documentation is installed in a directory that matches the current version number. The directory default location is the following:

```
<Infor Campaign Management installation directory >\ConfigFiles\Docs\<locale>
```

Some additional EM documentation and templates are located in the following default location:

`<Infor Campaign Management installation directory >\EM\docs`

For example, to view the installed documentation for Infor Campaign Management 10.1.0:

1. Navigate to **Start > Programs > Infor > 10.1.0 > Documentation > Infor Campaign Management > <locale>**.
2. Choose Documentation.

Installing Documentation on a Unix Host

1. Copy Supplemental_Files_< locale >_7100_unix.tar to a local directory.
2. Navigate to that directory and enter the following commands to unpack the PDF files (substituting the appropriate two-letter abbreviation for < locale >):

```
tar -xvf Supplemental_Files_<
locale
>_7100_unix.tar
```

```
gunzip Supplemental_<
locale
>tar.gz
```

```
tar -xvf Supplemental_<
locale
>.tar
```

Viewing Release Notes and Manuals Online

You can view the product documentation and Release Notes on any machine that has Acrobat Reader running.

When installing documentation on Windows, the installation wizard adds shortcuts to the Windows **Start** menu for easy viewing access. Documentation is installed in a directory that matches the current version number. The directory default location is: `C:\Program Files\Infor\Infor_Campaign_Management\ConfigFiles\Docs\<locale>`.

Some additional Infor Email Marketing documentation and templates are located in: `C:\Program Files\Infor\Infor_Campaign_Management\EM\docs`.

For example, to view the installed documentation for Infor Campaign Management, you can usually navigate to **Start > Programs > Infor > 10 > Documentation > Infor Campaign Management > <locale>** (depending upon where Infor Campaign Management is installed).

Choose either **Documentation** or **Release Notes** .

Printing This Document

Best print quality is achieved by printing this document with a PostScript driver. Other drivers may not reproduce screen shots accurately.

Contacting Customer Support

You may contact the Infor Customer Support center by submitting your incident via the web 24x7 at http://www.inforxtreme.com, or by placing a call during our scheduled business hours. For a complete listing of our support centers with web addresses and phone numbers, access our support site at http://www.inforxtreme.com.

Overview

2

Note: Do not copy and paste scripts, commands or code from this document. Line breaks and some other characters picked up when you copy from the PDF can cause errors in the pasted text.

This guide describes the procedures that you follow to configure and populate the data mart that is used by applications. This chapter gives an overview of functionality, architecture, and administration. provide you with a highly customizable, reliable, and robust framework for delivering a variety of enterprise-wide, customer-focused applications to users throughout your organization.

You can use Infor Campaign Management applications to review, analyze, and act on information that resides in a state-of-the-art data mart through a web-based interface. This thin-client interface runs on standard web browsers, requires no applets or plug-ins, and operates in a variety of supported languages and locales. Infor Campaign Management applications give users the following capabilities:

- Online analytical processing (OLAP)
- Data mining
- List management
- Campaign management

These powerful applications are deployed through Infor Campaign Management Server, an application server that allows them to work together in tightly integrated fashion. Depending on the source systems from which data is extracted, Infor Campaign Management applications can be applied to such diverse areas as:

- Sales reporting and analysis
- E-commerce management
- Supply-chain management
- Demographic analysis

The Infor Campaign Management Server treats individual application components as building blocks for creating sophisticated solutions to complex business problems. Navigation paths, called topics, link applications together in coordinated sequences and allow them to share data. Infor provides a growing number of prepackaged templates for topics that you can customize to meet the exact needs of your business.

The backbone of the system is a state-of-the-art data mart that includes extensions for list management and campaign analysis. You can use the data mart to collect data from a wide variety of data sources, including any data source with which you can establish an ODBC (Open Database Connectivity) connection. Typical source systems from which you can extract data include:

- Online transaction processing (OLTP) systems
- Online demographic databases
- Web logs
- Web-site traffic-analysis tools

Infor Campaign Management data marts reside on recognized, industry-standard relational database servers. Refer to the *Infor Campaign Management Installation Guide* for a list of supported database servers.

Infor Campaign Management System Architecture

As "Figure 1: Infor Campaign Management Architecture" on page 23 illustrates, Infor Campaign Management includes the following major components:

- EpiCenter Data Mart

  This data mart includes the following databases, which resides on different database servers:

  - The EpiMart database, which contains all of the data that is extracted from your source systems. Users choose from among the data and presentation elements that a web page displays to construct queries that read and report on data that resides in this database, which includes the following:
    - Standard data mart fact and dimension tables
    - Accelerators that improve response times for user queries, including indexes, aggregate tables, and integer maps for character columns
  - The EpiMeta database, which contains the metadata that defines:
    - The schema for your data mart
    - The extraction jobs that periodically update the fact and dimension tables of your data mart
    - The semantic rules by which updated data is merged into data mart tables
    - The scheduling information for extraction jobs and other periodic tasks
    - The set of accelerator tables and indexes to maintain
    - The specifications for each of the measures, attributes, filters, and web pages that you configure
    - The topics by which web pages are linked
    - The user and security information for your EpiCenter
    - The reports that users have saved
  - The EpiOp database, which contains the following:
    - Campaign history tables, which act as a secondary source system for campaign analysis
    - Lists, which include the results of previously executed queries
    - Extraction, scheduler, and user query logs

  Note: For Oracle installations, the term “database” in this manual refers to a schema.

- Source Systems

  Although not strictly part of the Infor Campaign Management system, source systems contain the raw data from which information is extracted and incorporated into your data mart. Source data can be extracted from any number of data sources.

- Admin Manager

  This graphical utility provides a complete, comprehensive, and easy-to-use interface for defining, configuring, and maintaining the Infor Campaign Management system, including the data mart, application components, and topics.

- Infor Campaign Management Server

  The Infor Campaign Management Server connects Infor Campaign Management applications to the data mart. This application server performs the following tasks:

  - Accepts the requests that each end user enters through the web page for an application
  - Constructs an optimized query for each request that takes advantage of available accelerators, cached results from previous queries, and so on
  - Forwards each query to the database server
  - Caches the results of each query and performs analytical calculations
  - Formats the results of each query for web-based display
  - Forwards those formatted results to the web browser

- J2EE Application Server

  The J2EE Application Server routes URL requests that your web server receives to the Infor Campaign Management Server for processing.

- EpiChannel Extraction, Transformation, and Loading (ETL) Utility

  The EpiChannel utility initiates the extraction jobs that perform the following tasks:

  - Pull data from individual source systems, possibly cleanse it, and insert it into staging tables
  - Apply semantic transformations to ensure that data is updated in a manner that is consistent with your business rules
  - Incorporate the data into your data mart
  - Build accelerators such as aggregate tables and indexes, which are used to improve the response time for reports, as well as integer maps and fact samples, which are used to construct lists and campaigns

  EpiChannel also allows you to call third-party programs, such as data-cleansing tools and other utilities, as part of your extraction process.

- EpiChannel Service or Daemon

  The EpiChannel service or daemon allows the EpiChannel utility to be invoked as needed by the scheduler service.

- Scheduler Service

  The Scheduler initiates asynchronous or recurring actions such as extraction jobs, long-running reports, and campaigns.

The combination of data mart, source systems, Admin Manager, EpiChannel, and related utilities is often referred to as the back end of Infor Campaign Management. The combination of Infor Campaign Management Server, web server and J2EE application server, web browser, and web-based applications is often referred to as the front end.

Within the back end, periodic extraction jobs collect data from source databases through native or ODBC connections. The system typically accesses source data in a read-only fashion, so no changes to the content of existing source systems are necessary. However, you can include housekeeping tasks that run on a source system within an extraction job if you choose to do so.

At the front end, each user opens a web page on any computer that supports a browser and then follows a sequence of preconfigured links to reach a desired application. The user completes a form that specifies the contents of a report and then clicks a button to initiate a query. The type of query that is issued, and the analytical calculations that the Infor Campaign Management Server performs on the results, depend on the application (web page) from which the query originates.

The web browser passes the user's request to the web server, which in turn calls the J2EE application server to route the request to the Infor Campaign Management Server application server. The Infor Campaign Management Server optimizes the query by selecting appropriate aggregate and sample tables and indexes, then passes the optimized query to the data mart over a JDBC connection. When the data mart returns the result over that same connection, the Infor Campaign Management Server instantiates the appropriate application-specific Java classes to perform analytical calculations, and then formats the finished results for presentation in HTML format. The Infor Campaign Management Server then forwards these formatted results to the requesting user's web browser for display.

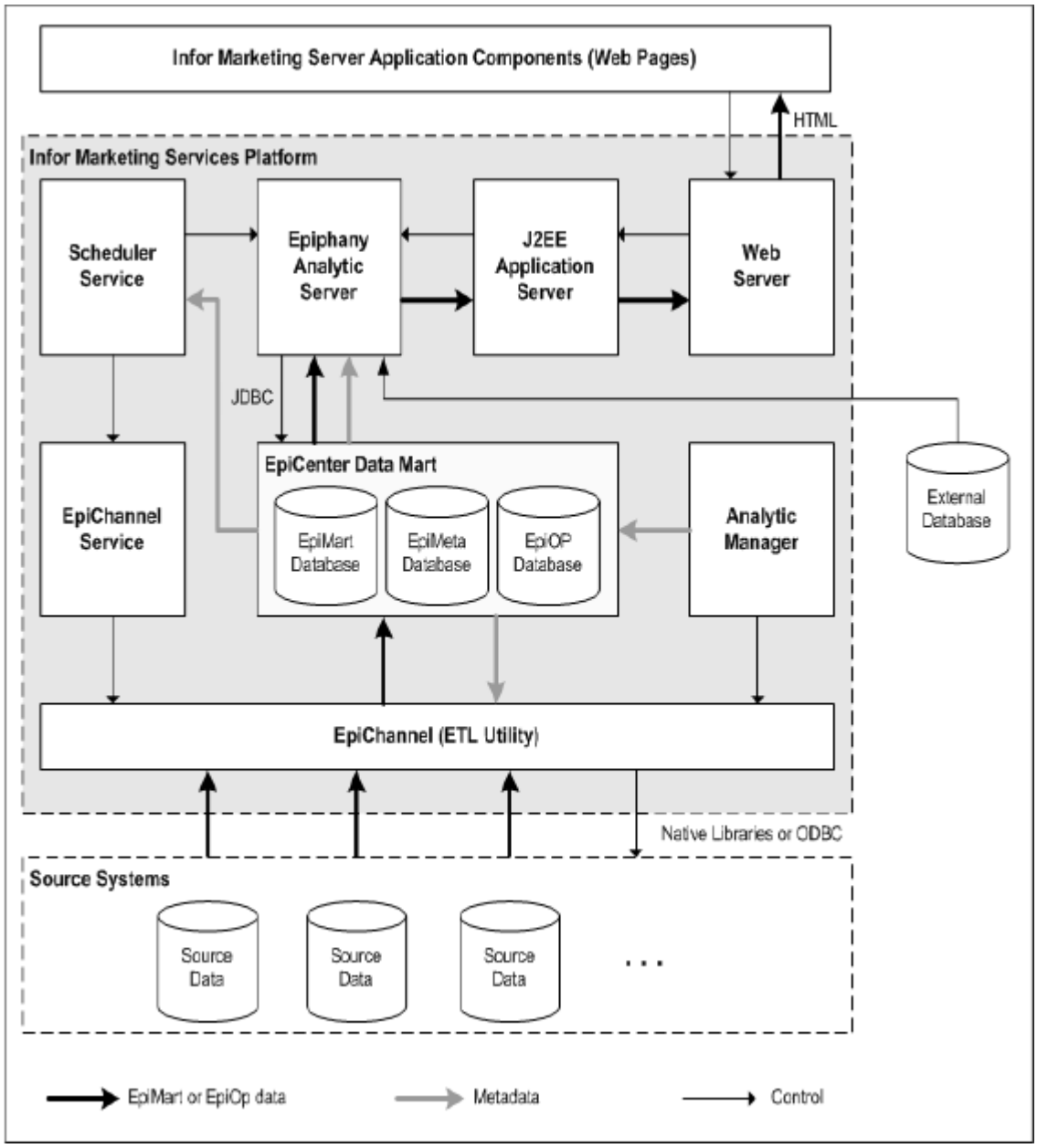


Figure 1: Infor Campaign Management Architecture

Infor Campaign Management Metadata

All of the control information for an EpiCenter data mart is stored in a single metadata repository, the EpiMeta database. This database uses a relational model that includes numerous tables and referential

integrity constraints. The use of a single metadata repository (EpiMeta) ensures that all components of the system receive notice of a change simultaneously.

Because the database defines the structure of an datamart and the applications that operate on that data, includes export/import commands that allow you to back up metadata, or to transfer subsets of metadata between datamarts.

In traditional client/server application environments, changes to the schema of a database table can adversely affect programs that operate on that table. The goal of the adaptive architecture is to allow on-the-fly changes to the datamart schema while preserving the proper operation of applications.

For example, the semantic instances that apply business rules to extracted data are SQL programs that must be changed in response to schema changes. References to tables and columns are parameterized within these instances so that can update them at execution with current schema information. applications and other system components also refer to current metadata for information about the schema of the datamart, the configuration of applications, and the structure of topics. See "The Scrutiny Debugging Tool" on page 386 for more information.

Infor Campaign Management Utilities

The Infor Campaign Management system provides the following utilities for configuring the data mart, extracting source-system data, and deploying applications:

- Admin Manager
- EpiChannel
- The Infor Campaign Management Server (application server) service or daemon
- The Scheduler service or daemon

The sections that follow describe each of these utilities.

Admin Manager

The Admin Manager is a graphical interface that maintains all of the metadata for your Infor Campaign Management system and applications. A scripting version of this utility allows you to perform the same administrative tasks within command scripts.

EpiChannel

EpiChannel reads schema, extraction, and semantic metadata and writes logging data about each extraction job. EpiChannel provides the following:

- A macro language that you can use to create parameterized SQL extraction programs

- An extensive error checking and reporting functionality
- An extensive logging of extraction jobs
- The ability to automatically restart extraction jobs that encounter transitory problems
- The ability to create aggregate tables that greatly reduce response times for the queries that use them
- The ability to create accelerators, including integer maps and samples, which are essential for generating lists and campaigns

Infor Campaign Management Server

Infor Campaign Management Server is an application server that supports Infor Campaign Management applications and connects those applications to the data mart. Infor Campaign Management Server handles user requests by doing the following:

- Forwarding optimized queries to the database server
- Caching query results
- Instantiating appropriate Java classes to perform application-specific calculations on those results
- Forwarding the final results for web-based display

The Infor Campaign Management Server also provides a web-based interface for monitoring its operations and refreshing its state as necessitated by changes to metadata.

Scheduler

The Infor Campaign Management scheduler allows administrators to schedule individual and recurring extraction jobs. It also allows users to schedule long-running reports and campaigns for execution during off-peak hours. Users can also schedule recurring reports and campaigns.

International Language Support

Localized versions of Infor Campaign Management allow you to configure applications in which users view and generate reports entirely in their local language. The browser through which a user views an application must be set to use a code page that is capable of rendering characters in that language. In addition, the `language_locale_key` user preference must be set to a language that is supported within the data mart. See “Default User Preferences,” in chapter 6 of the *Infor Campaign Management Topic Implementation Guide*.

Infor Campaign Management provides support for localized applications through the following components:

- The EpiMart database

Data values within data mart tables can be specified in any supported language and locale. However, table and column names within the data mart are restricted to the ASCII character set.

A single EpiCenter data mart can contain data in multiple languages and locales. Multilingual access to a single data mart is also supported. Browsers that support the appropriate code page for that language and locale can display EpiMart data.

- The EpiMeta database

  This database must use the same code page for localized characters as the EpiMart database. User-visible labels and prompts that reside in metadata are stored in multiple languages. Descriptive commentaries for specific objects, which are viewed only by implementors, are also stored in metadata in multiple languages. Consequently you can define an object in one language, have implementors modify that object in a second language, and allow end-users to view that object in a third language.

- The EpiOp database

  User-generated data in this database appears in the same language, locale, and code page as that of data mart values.

- EpiChannel extraction commands

  The following elements of an SQL extraction command can be specified in any supported language and locale:

  - References to source-system tables and columns
  - References to data values
  - Literal (single-quoted) strings

- Admin Manager

  The graphical interface for Admin Manager is fully internationalized.

  If the default code page of Admin Manager's host differs from that of the database server in which the data mart resides, Admin Manager automatically converts localized data that is entered or displayed to the appropriate code page. That is, when Admin Manager displays localized data from the EpiMeta database, that data is converted to the appropriate code page for Admin Manager's host. When data is entered into a localized field in Admin Manager, that data is automatically converted to the appropriate code page for the EpiMeta database before it is stored there.

  Note: The code page that is used on Admin Manager's host must be capable of rendering characters in the EpiMart. Otherwise, those characters cannot be rendered correctly.

- Infor Campaign Management Server

  The Infor Campaign Management Server displays localized versions of log messages, prompts, error messages, and other text elements. However, exceptions are always displayed in English with the ASCII character set.

EpiCenter Schema Design

3

This chapter introduces the Infor Campaign Management database schema. A database schema is a structure for organizing data into tables. The schema of an EpiCenter (the Infor Campaign Management data mart) allows users to access information rapidly, in a form that allows them to understand and take appropriate action. The EpiCenter schema allows users to generate reports, charts, lists, and campaigns. Each data mart requires its own schema, or table layout, which depends on the data that you choose to include.

The EpiCenter Database Schema

The EpiCenter schema uses a dimensional data warehouse model. A data warehouse transforms the raw data from an organization's source-system databases into a star-schema format that is optimized for end-user ad hoc query and analysis. Data in the data mart conforms to the business model for your organization. Extraction jobs ensure that this data is consistent, reusable, and flexible. You can sort and group the data by any measure that your business uses. A data warehouse also provides the tools that are needed to query, analyze, and publish this data.

For more information on the theory and practice of data warehousing, see *The Data Warehouse Toolkit* by Ralph Kimball.

The Dimensional Data Model

You can visualize a dimensional data warehouse as a cube filled with data about measurable facts, such as the amount of purchases, orders, or other measurable quantities. Each axis, or dimension , of the cube represents a type of descriptive information about each fact, such as the name of a purchaser, the date on which a transaction occurred, or the location at which a return took place. Fact data is distributed throughout the cube based on the associated descriptive information, with each data point being placed so that the values along the axes correspond to the descriptive information associated with that data point.

Figuratively speaking, you can take a cross-section (or slice) at a point along any dimension of the cube to get a report of the data that pertains to a particular value within that dimension. For example,

if the cube containing information about orders has the dimensions of Customer, Territory, and Time, you can slice through the cube along the Time dimension to find order information for a specific date. If you slice the cube by Territory and Time, you can see orders for a given region in a given period. If you slice along the Customer dimension, you can see the order information for a given customer.

Figure 3-1: The Dimensional Data Model

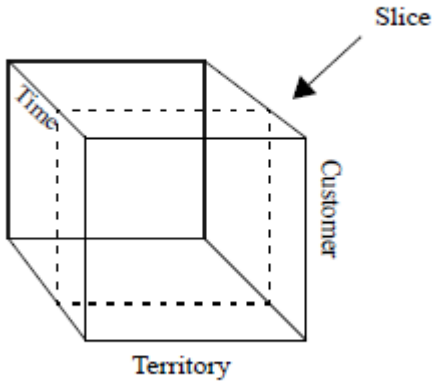


Figure 2: The Dimensional Data Model

The Star Schema

The EpiCenter represents a dimensional data warehouse with database tables organized in a star schema. (See "Figure 3: A Standard Star Schema" on page 29 below.) At the center of the star schema is a fact table that contains measured data. Radiating outward from the fact table like the points of a star are multiple dimension tables . Dimension tables contain attribute data, such as the names of customers and territories.

Figure 3-2: A Standard Star Schema

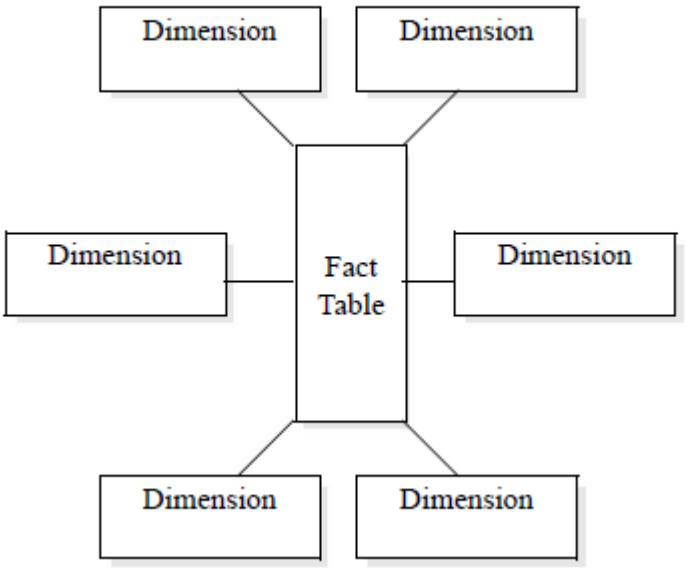


Figure 3: A Standard Star Schema

In a standard star schema, the fact table is connected, or joined , to each of the dimension tables, but the dimension tables are connected only to the fact table. This schema differs from that of conventional relational databases, in which many tables are interjoined.

An Infor Campaign Management star schema consists of several interjoined stars. (See "Figure 4: The Infor Campaign Management Star Schema" on page 30.) At the center of each star is a fact table, and there is a global collection of dimension tables that can be shared by stars. By sharing dimensions, Infor Campaign Management can generate reports from data that comes from several different fact tables.

For example, an Expense fact table and a Sales fact table might contain information broken down by date and product line. If the data in both of these fact tables is accessible, an end user can create a report that shows relationships between expenses and sales based on the date and product line attributes.

Figure 3-3: The Infor Marketing Star Schema

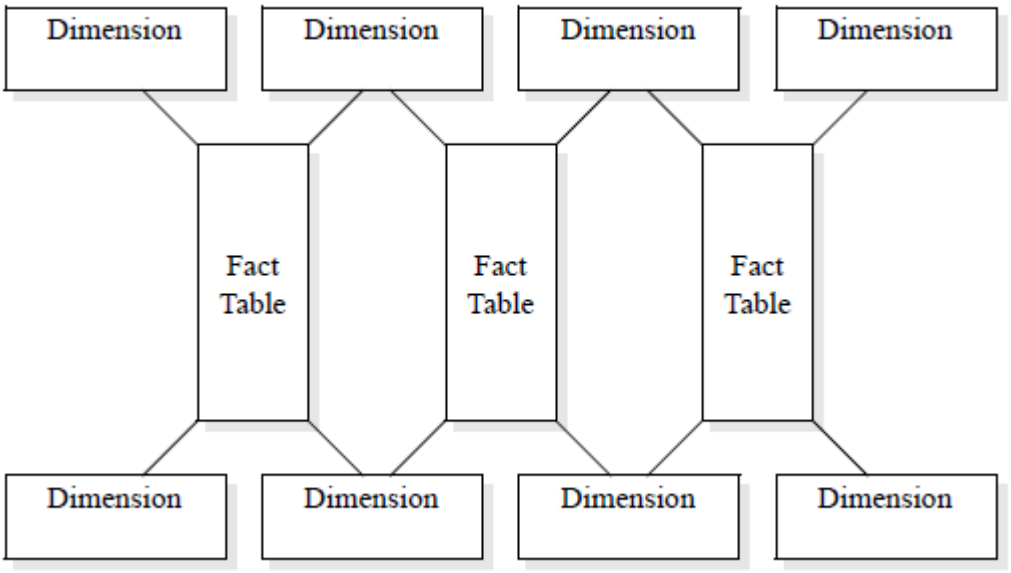


Figure 4: The Infor Campaign Management Star Schema

Note: The Date Dimension, a built-in dimension that handles time, automatically joins to other dimensions if those dimensions include data about dates and times. No other dimension can join to another dimension. See "Snowflaked Base Dimensions" on page 36 for information.

Infor Campaign Management allows you to regenerate an EpiCenter when your business model changes. Instead of replacing the entire data warehouse, you can simply modify the database schema and update your EpiCenter. The Infor Campaign Management system also facilitates the creation of new EpiCenters for an organization, since you can easily import data from an existing EpiCenter into another EpiCenter.

The EpiCenter

An EpiCenter is composed of EpiMeta, EpiMart, and EpiOp databases. The EpiMart database contains all of the actual data mart tables, and the EpiMeta database contains all of the Infor Campaign Management metadata tables. Metadata is information about data and its organization, but not the data itself. The EpiMeta database defines the schema for the EpiMart tables that ultimately contain the actual extracted, organized data (such as customer, product, and order data). It also defines the navigational paths that an end user can follow when working within a set of related web pages (known as a topic), and determines what data from a web page is persistent from one point of this path to another. (See the *Topic Implementation Guide* for a description of navigation within a topic.) The EpiOp database contains operational data generated by the Infor Campaign Management system. An EpiCenter is an EpiMeta database with its associated internal links to EpiMart and EpiOp databases.

The EpiMart consists of fact, dimension, and staging tables and accelerators. The fact and dimension tables contain actual data mart data. Staging tables, which are discussed in "Database Extraction Overview" on page 169 are the first entry point of raw data from the source systems into the EpiMart. Staging tables are an interim stop before data reaches EpiMart tables.

To facilitate the building of EpiCenters, Infor provides Admin Manager, a Microsoft Windows application with an Explorer-like hierarchical structure. You use Admin Manager's graphical user interface to define the schema for the EpiMart tables, configure web pages, define the extraction process, set system security, and instantiate end-user navigation paths for topics from pre-defined Infor Campaign Management topic templates.

Figure 3-4: Infor Marketing Manager Window

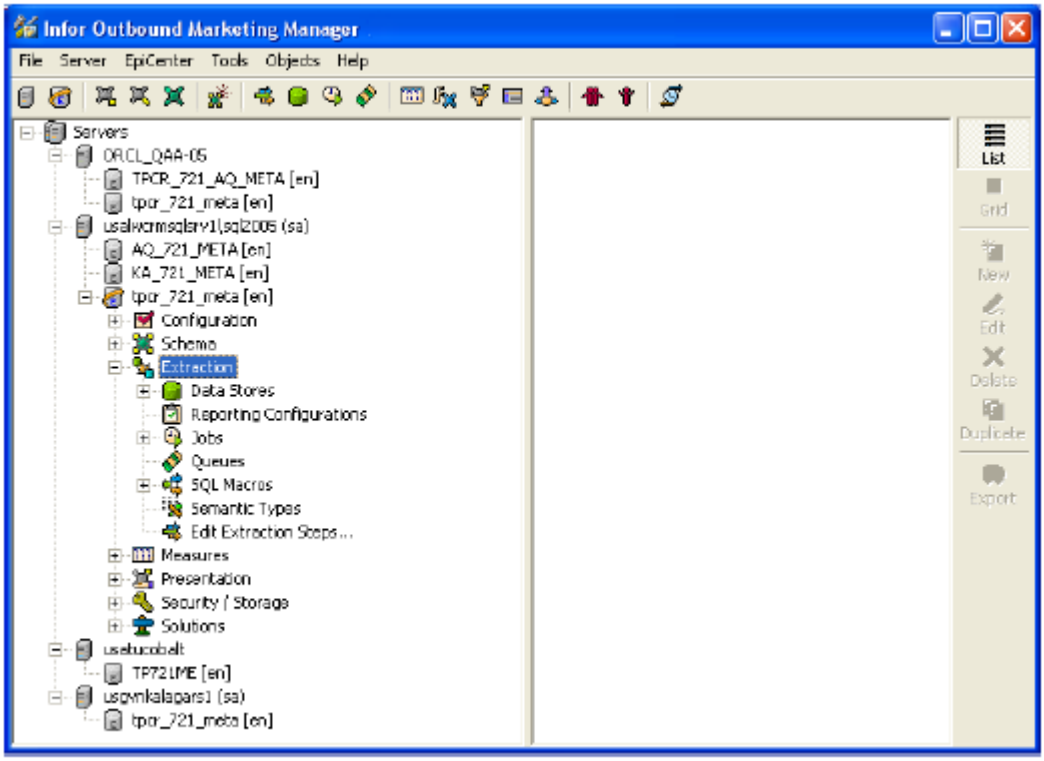


Figure 5: Admin Manager Window

Dimension Tables

A base dimension table consists of information for elements, such as product, customer, or company, that are referenced in fact data. Each column of a dimension table corresponds to some characteristic of such an element, such as customer age or customer income. The actual values in a dimension table are extracted from your source systems.

The Infor Campaign Management schema allows dimensions to be shared by all of the fact tables within an EpiCenter. Each base dimension table can be referenced by multiple fact tables, or multiple times by the same fact table. These references are called dimension roles . Dimension roles are global objects that can be used by several fact tables. Whereas the base dimension tables are physical database tables, the dimension role is a logical entity that defines the meaning of the dimensions to the EpiCenter.

Base Dimensions

A typical star schema has multiple dimensions associated with a fact table. For example, an Order fact table of 20 million rows have the dimensions Product (1000 rows), Customer (3000 rows), and Date (365 rows). In theory, you can construct a star schema that has only one dimension; that is, a supra-dimension that contains all three dimensions. The resulting dimension table, however, needs one row for every possible combination of dimension values that appear in the fact table.

Why not build a single dimension table in this case? The Product, Customer, and Date dimension values are independent of one another. For example, these three dimension tables might have 1000, 3000, and 365 rows, respectively. However, there are more than one billion possible combinations of these values. If you have a single dimension table, then that table does not require a row for every possible combination of dimension values, but it must have a row for every such combination in the fact table. Since few customers are likely to order a given product more than once in a single day, this dimension table is likely to have almost as many rows as the fact table.

The only trade-off involved in using multiple dimensions is that each dimension has an associated dimension role key in the fact table that is a 4-byte integer (for a mapped dimension), and these bytes accumulate. In the example above, the extra dimension-role keys are worth the extra space.

For example, with a fact table of 2 million rows, a single supra-dimension might have more than 1 million rows. If each row of the smaller dimension tables takes up about 100 bytes, then each row of the supra-dimension takes up about 300 bytes. Thus, the supra-dimension takes up about 300 million bytes. On the other hand, the three smaller dimension tables take up a total of less than 500,000 bytes. The two extra foreign keys in the fact table (necessitated by the two additional dimension tables) take up 8 bytes per row, for a total of 8 million bytes. The grand total for three separate dimensions is therefore less than 8.5 million. In other words, in this example, the single supra-dimension takes up more than 35 times as much space as three separate dimensions.

In addition to the space savings, using multiple dimensions results in significant performance improvements. In the example above, the three small dimension tables, with a total of under 5000 rows, can be scanned much more quickly than a single dimension table with a million rows. This speedup results in significant improvements in both query-response time and extraction time.

Sometimes combining two small, independent dimensions can create a sense. For example, assume that SalesPerson has ten distinct values in the Order fact table, and Promotion has 20 distinct values. Even if these two dimensions are completely independent, building a composite dimension with several hundred rows, which is very small, can save a role key in the fact table.

Attributes and Granularity of Data

Attributes refer to columns within a dimension table (or degenerate dimensions within a fact table). In the EpiCenter, facts are usually quantifiable measurements expressed as numbers. Attributes of dimensions are usually character strings. Some attributes, such as Customer Name, are free-text descriptions, but most have a small number of possible values.

For example, a Sales Territory dimension table can contain a `sales_territory` key that uniquely identifies a sales territory and a **region** text field that describes the sales regions (for example, Eastern, Western, or Southern) in which the sales territory is located. This dimension table can also contain attributes for subdivisions of a region, such as state and county (this assumes that no sales territory includes more than one county). In this way, there can be a hierarchical relationship within the dimension table, in which the county attribute rolls up (aggregates) into state, which then rolls up into **region** .

When implementing a database schema, consider the granularity of data, or level of detail, to which end users drill down . (Drilling down refers to applying the same query criteria to a subset of a previous query's results to obtain more specificity, usually by filtering on selected attributes.) In the Sales Territory table, a user can drill down from sales region to state, and then to sales territory. The value of the `sales_territory` field is the finest level of granularity that is available in this dimension table.

In the example in the table below, the granularity of data becomes more specific as one moves from Region to State to City. If queries are prepared for the set of data at the Region level, but not at the City level, the granularity of data can be at the higher level. The data at a higher level of granularity can reduce the size of a dimension table.

Table 1: Example Data Granularity

Region	State	City
Western	CA	Oakland
Western	CA	Petrolia
Western	OR	Portland

Dimension Versioning

A dimension may contain multiple versions of the data for a single dimension element. For example, you might add a new dimension row for a customer if demographic information for that customer changes. By adding a new row with the changed data, older facts can continue to refer to the row that contains older demographic data. This allows you to analyze fact data based on the demographic information that is current at the time that the fact occurred.

If you choose not to record multiple versions of data for a single dimension element, you can specify that a dimension is non-versioned. The use of a non-versioned dimension can reduce the width of fact tables that are joined to that dimension and can result in faster campaign query performance.

Primary and Secondary Dimension Columns

If desired, the columns of a dimension can be divided into primary and secondary columns.

Generally, primary dimension columns are the columns that are of interest for data analysis, such as age or product category. You can choose to build dimension aggregates using primary dimension columns in order to speed up analytic queries.

Secondary columns usually contain information that can be useful for such things as list or campaign output, but that is generally not used in analytic queries. Secondary columns can contain data like street addresses or SKU numbers. In most cases, secondary dimension columns are not used in dimension aggregates.

Semantic instances can use different strategies to update the primary and secondary columns. For example, you can define a semantic type that updates a dimension row when only values in secondary columns change, but that creates a new version of a dimension row when a value in a primary column changes.

UNKNOWN Dimension Values

Null values are not allowed in dimension columns. Each dimension column has a default value, called the UNKNOWN value, that is normally used to indicate that a value is not known for that column. By default, the string `'UNKNOWN'` or the integer value 0 is used for the `UNKNOWN` dimension value. You can use Admin Manager to change the **UNKNOWN** value for a dimension column.

Cardinality of Dimension Columns

Some dimension columns have a low number of possible values. For example, a dimension column that stores the U.S. state in which an individual lives has only 50 possible values, even though the dimension itself may have millions of rows.

If you indicate that a dimension column has low cardinality then the EpiCenter can use that information to optimize the dimension table structure. You can also choose to integer-map a low cardinality dimension column.

Integer Mapping

If a low cardinality dimension column has values that take up significant amounts of storage (for example, if the data type is `VARCHAR_50`), then you can significantly decrease the size of the dimension table by integer mapping those values.

When you integer map a dimension column, the extraction machinery associates a unique integer value to each value of the dimension column. The small integer values are then stored in the actual dimension table, and a separate mapping table records the association between these integer values

and the original values. At query time, the query machinery uses the mapping table to recover the original dimension-column values.

For example, if you have a dimension that stores the name of a U.S. state in a column, then you can save significant amounts of space by integer mapping that column. On the other hand, if this column only stored the two-letter code for each state, then integer mapping does not result in significant space savings.

All integer-mapped dimension columns must be marked as being low cardinality. If the cardinality of an integer mapped dimension column grows larger than the `IntmapMaxValuesWarn` configuration setting (located in the Configuration dialog box under **Behavior > EpiChannel**), Infor recommends that you deselect the **Perform Integer Mapping** check box for the dimension column and regenerate the schema in order to undo the integer mapping for that column.

Since the structure of the dimension table must be changed when generating the schema after changing the integer mapping option, schema generation may take some time to complete. You may wish to increase the values of `IntmapMaxValuesWarn` and `IntmapMaxValues` until you are able to regenerate the schema.

DB2 restrict integer mapping to columns that are less than 1024 bytes in length. SQL Server restricts integer mapping to columns that are less than 900 bytes in length.

The Transtype Dimension

Each row of a fact table has an associated transaction type. The transaction type allows you to store facts about different kinds of events in the same fact table. For example, you can store information about both sales and returns by assigning one transaction type to sales facts and another transaction type to returns.

The Transtype (transaction type) dimension is a built-in base dimension that is common to all fact tables. By looking at only those facts in a fact table that have a given transaction type, you can in effect look at a “slice” of the fact table. The Transtype dimension consists of all transaction types that is defined for your EpiCenter. Thus, you can use the Transtype dimension to distinguish among different rows in a single fact table based on their transaction type, such as shipping transactions versus booking transactions, and bookings versus booked returns.

The Date Dimension and Uniform Treatment of Time

The built-in **Date** dimension is common to all fact tables. To ensure consistent treatment of time, the EpiCenter uses a single date dimension throughout the system.

In many Enterprise Relationship Management (ERM) applications, the time that a fact occurs is a crucial dimension. Date arithmetic can become complicated, however, because of the non-uniformity of the calendar with respect to such factors as the number of days in the month, leap years, and weekly overlaps with months.

The EpiCenter provides the special **Date** dimension to simplify and unify the treatment of dates. Each row in the **Date** dimension table represents a single day and the attributes associated with that day. You use Admin Manager to populate the date dimension with the parameters that makes sense for your implementation, generally when you generate your EpiCenter schema. When the **Date** dimension is populated, each date is assigned values for such attributes as the week, month, quarter, and year to which the date belongs.

Every fact table in your EpiMart automatically contains a foreign key to this special date dimension. By joining the fact table to the date dimension, Infor Campaign Management applications are able to answer questions like the following:

- Which facts occurred this quarter?
- What is my weekly backlog for the year?
- What products were bought last month?

Without a **Date** dimension, such queries are needed to be formulated in terms of unwieldy date-arithmetic calculations. Performing date arithmetic for each query of this kind can significantly slow system response time. Using the **Date** dimension improves performance.

Infor Campaign Management allows for specification of Calendar fiscal quarters, as well as 13-week manufacturing calendars (4-4-5 calendars), in which a quarter always starts on the same day of the week. The **Date** dimension can also be configured to specify which weekdays start and end a business week (for example, Sunday through Saturday versus Monday through Sunday).

Snowflaked Base Dimensions

Base dimensions that include date columns can also join to the Date dimension in a snowflake configuration. Snowflaking occurs when two dimensions in a star schema join directly to one another. Snowflaking is only supported in Infor Campaign Management when the joined dimension is the Date dimension. Allowing dimensions to join with the Date dimension allows users to create reports such as total sales by customer birth month, or answer queries such as, “How many customers are having birthday in the next two months?”

Date dimension snowflaking occurs automatically whenever you define a base dimension column with a **SMALLDATE** physical type.

Partitioning Oracle Base Dimensions

If you are using an Oracle database server with the Oracle Partitioning option, you can choose to partition your dimension tables. Partitioning a dimension table can improve extraction performance by allowing parallel update and delete statements.

Note: When you partition a dimension table or index, only the dimension base and thin tables are partitioned. Dimension mapping tables are not partitioned.

Partitioning DB2 Base Dimensions

If you are using an IBM DB2 database server, you can choose to partition your dimension tables. When you create or edit a base dimension, you are presented with the following partitioning options at the bottom of the **Dimension** dialog box.

Figure 3-5: Dimension Dialog Box (Object Properties Tab)

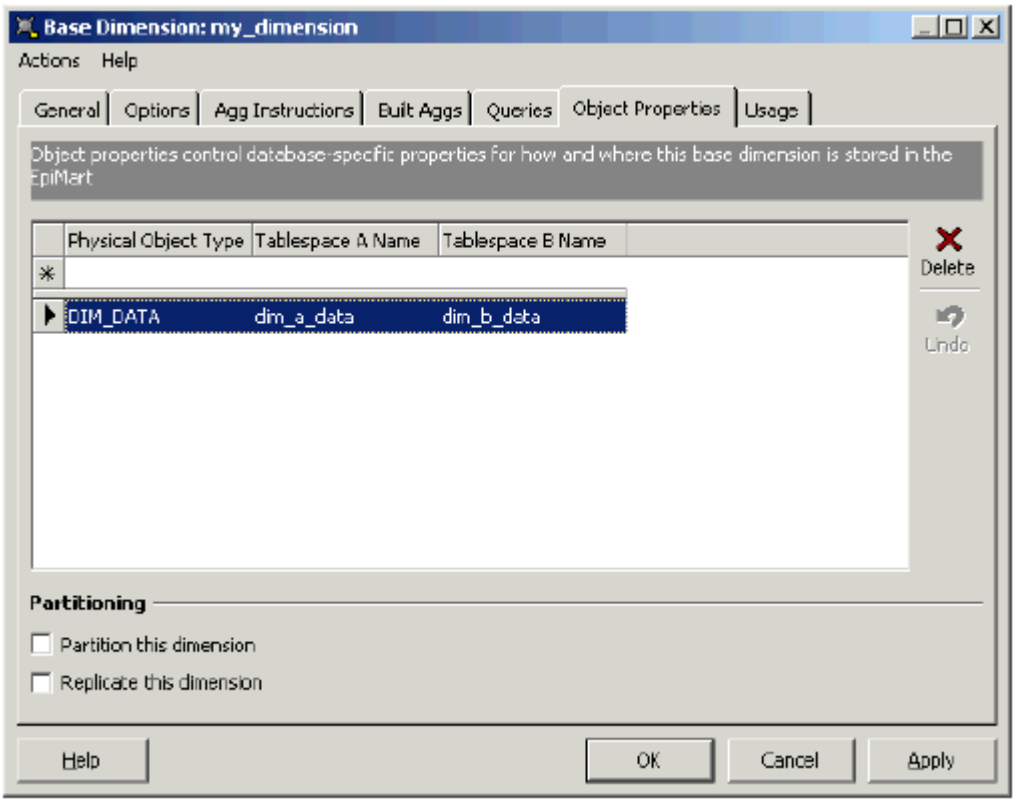


Figure 6: Dimension Dialog Box (Object Properties Tab)

You can choose to partition or replicate the selected base dimension by selecting the appropriate option. Partitioning in DB2 is done through hashing, but not range partitioning. See the DB2 documentation for more information.

Replication Considerations: Generally replicate those tables that are small and are used with every partition of a large fact or dimension table. Good choices include date and transtype tables. See "DB2 Replication Object Types" on page 521 for information on the physical object types supplied for replication.

Partition Considerations: Generally partition large fact and dimension tables that are frequently joined and that are partitioned and accessed using the same hashed key. For example, if you have a dimension of `individuals` and a fact table of `orders`, these are ideal candidates for partitioning as the individuals and their orders are found in the same partition using the same hashed key.

Note: Partitioned objects must be placed in multi-node tablespaces.

If you partition the dimension (DIM_DATA) or fact (FACT_DATA) tables and intend to replicate the mapping tables, make sure the appropriate replicated dimension mapping (DIM_MAP_REPL_DATA) and/or integer mapping (INTMAP_REPL_DATA) tables are replicated into the tablespace containing the dimension or fact tables.

Fact Tables

Fact tables are physical database tables that contain fact columns and dimension-role foreign keys. A fact column is a single column of the fact table in which the values are numeric representations of business data, such as net_price. In an EpiCenter, a measure is a formula for a business calculation that you define over one or more fact or dimension columns.

A row in a fact table represents an association among the set of values in that row. For example, a fact-table row can contain data for a single transaction. As shown in "Figure 7: Sample Row in a Fact Table" on page 38, such a fact table row contain values for a specific customer number, a product number, a territory number, a marketing program, and a date. These values are all foreign keys that point to base dimension tables for their data. In addition, there can be fact columns for such data as the number of units sold and the price per unit.

This row of the fact table represents a single transaction (such as an order) involving the specified quantity of a single product at the specified unit price. The dimension-table foreign keys indicate the customer, product, territory, marketing program, and date for that transaction. The fact table can have millions of such rows of data.

Figure 3-6: Sample Row in a Fact Table

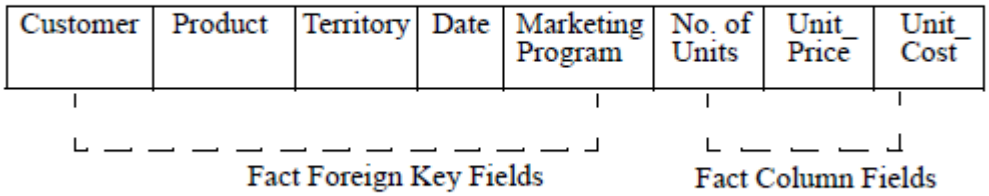


Figure 7: Sample Row in a Fact Table

Dimension Roles

A fact table can have as many foreign keys to a single base dimension table as the business model requires. (A foreign key is a reference from one table to another table.) For example, a fact table for Orders can have foreign keys for **Customer_Bill-to** and **Customer_Ship-to** that point to the **Customer** base dimension table. These foreign keys are defined in metadata by means of dimension roles.

A row in a fact table have several foreign keys that point to the same base dimension table, but that are used in different roles. For example, a fact table of auction sales can refer to both the buyer and the seller for each transaction. Since any given customer can be a buyer in some transactions and a seller in other transactions, you can have a single Customer dimension table. You can then have buyer and seller dimension roles that allow a single fact table to refer to the Customer dimension table with respect to these two different roles.

You can perform arithmetic operations on fact-column values based on these dimension foreign keys. For example, you can add the values in a fact column for every element of a dimension (that is, for a given dimension element, all fact-column values in rows with foreign keys that refer to that dimension element are added). These totals can then be presented in report and chart format.

The organization of the Infor Campaign Management database allows users to create flexible queries. For example, a user can query the EpiMart for the names of customers that attends a particular trade show to determine the dollar amount of orders these attendees purchased as a result of this show.

Degenerate Dimensions

A degenerate dimension is a single column in a fact table that stores textual information. A degenerate dimension is similar to a dimension table with a single column. Rather than storing the values for the single dimension column in a separate table and storing foreign keys to that table in the fact table, the dimension-column values are stored directly in the fact table.

When defining degenerate dimensions, keep in mind that degenerate dimensions cannot be included in aggregates (See "Aggregate Building" on page 43), so end-user queries that refer to degenerate dimensions cannot take advantage of the performance improvement provided by aggregates.

Uniform Transactional Data

A transaction represents an event in time. All data in an EpiMart fact table are stored in transactional format. For certain business entities, this format makes intuitive sense, such as in the case of an invoice in which a record in EpiMart represents the shipment of a product to a customer at a certain point in time. For the most part, this event cannot be changed; it simply happened and is stored as such in an EpiMart fact table.

Other business facts are not initially presented as transactions. A customer can call and order 10 units of a product. This is specified as a transaction with quantity 10 for product P1 to customer C1, as displayed in this table:

Table 2: A Customer Order

Order Number	Customer	Product	Date	Quantity
2X5397	C1	P1	6/30/1999	10

However, if the customer calls back the next day and changes the order to 15 units, then it is not correct to enter a separate transaction of 15, since those two transactions are added to give a total quantity of 25. Since Infor Campaign Management treat all facts as transactions, the revised order must be converted to a transaction. Infor Campaign Management extraction machinery can automatically create a difference transaction to ensure that the new data is specified in the proper form. In the example above, the extraction machinery enters the difference of 5 as a new transaction on the second day, as shown in this table:

Table 3: A Revised Customer Order

Order Number	Customer	Product	Date	Quantity
2X5397	C1	P1	6/30/1999	10
2X5397	C1	P1	7/1/1999	5

Now the total quantity for the order correctly adds up to 15. In addition, all parts of the order are assigned to the correct date. In the example in the table below, the first 10 units are included in the second calendar quarter and the additional 5 units are included in the third calendar quarter.

Similarly, in the case of Inventory, Infor Campaign Management can track changes in inventory as transactions instead of restating the reported inventory. For example, Customer C1 can report inventory for Product P1 as displayed in "Reported Inventory in Source System" on page 40:

Table 4: Reported Inventory in Source System

Date	Quantity On Hand
6/1/1999	10
6/8/1999	12
6/15/1999	5
7/3/1999	14

In the Infor Campaign Management fact table the same information can be represented transactionally as shown in "Transactional Version of Inventory" on page 40:

Table 5: Transactional Version of Inventory

Customer	Product	Date	Change In Inventory
C1	P1	6/1/1999	10
C1	P1	6/8/1999	2
C1	P1	6/15/1999	-7
C1	P1	7/3/1999	9

When an end user asks for a report of inventory by week, these transactions can be added to give the original data as it is reported by the source system. To reassemble the data in this way, you can define

a backlog measure that gives the accumulated values at any point in time. (See “Measures” in chapter 5 of the Topic Implementation Guide for a discussion of backlog measures.)

Why go to the trouble of disassembling orders and inventories into transactions, only to reassemble the previous format at output time? The reason is that transactions are additive, which means that the transactions can be recombined along arbitrary dimensional boundaries.

The way in which Inventory data is reported above can easily answer queries of the form “Inventory by week. ” This data, however, cannot easily be used to answer queries by month or year without some external knowledge of which week is the end of the month. Infor Campaign Management transactional storage, together with the built-in **Date** dimension, makes it easy to answer such queries. In "Transactional Version of Inventory" on page 40, the inventory for the month of June can be calculated by adding up all of the transactions that occur between 6/1 and 6/30, yielding an ending inventory of 10 + 2 – 7 = 5.

Note: All source-system fact data must be created into transactions. For state-like facts, this completes by applying an appropriate semantic instance. Keep this in mind when you author an Infor Campaign Management extraction command or choose a semantic type.

Partitioning an Oracle Fact Table

If you are using an Oracle database server with the Oracle Partitioning option, you can choose to implement historical partitions on your fact tables. You can partition fact data by day, week, month, quarter, or year.

Partitioning a fact table in this way can improve extraction performance by placing historical data, which tends to change infrequently, in a different partition from the current data, which often changes with each extraction. Fact-table partitioning also enables you to roll off old data that is no longer needed.

Facts with missing dates are assigned the UNKNOWN date value. (The UNKNOWN date is a special row in the Date dimension. See "The UNKNOWN Dimension Row" on page 182 for more information on the use of the UNKNOWN row.) By default, the UNKNOWN date value is 01/01/1900.

Oracle places all data with a date prior to the specified partition start date into the earliest partition, so facts with the UNKNOWN date typically appear in the earliest partition. Infor suggests that you set your partition start date to a value that ensures that no legitimate date values (that is, date values other than the UNKNOWN date) fall into the earliest partition. That way, if you wish to roll off old data, you can use the TRUNCATE PARTITION operation rather then the much slower DELETE FROM operation to remove old data.

For partitioned fact tables that do not have an index defined on the date dimension role, setting the /Optimization/Query/UsePartitionedFactDateRangeFilter configuration setting to 1 in Admin Manager, can improve the performance of queries on those tables. If partitioned fact tables have indexes defined on the date dimension role, then Infor recommends leaving this setting at the default value of 0.

Note: Create database partitions using Infor only. Do not use Oracle to partition the database.

Partitioning a DB2 Fact Table

If you are using an IBM DB2 database server, you can choose to partition your fact tables. When you create or edit a base fact, you are presented with the following partitioning options at the bottom of the **Fact Table** dialog box ("Figure 8: Fact Table Dialog Box" on page 42).

Figure 3-7: Fact Table Dialog Box

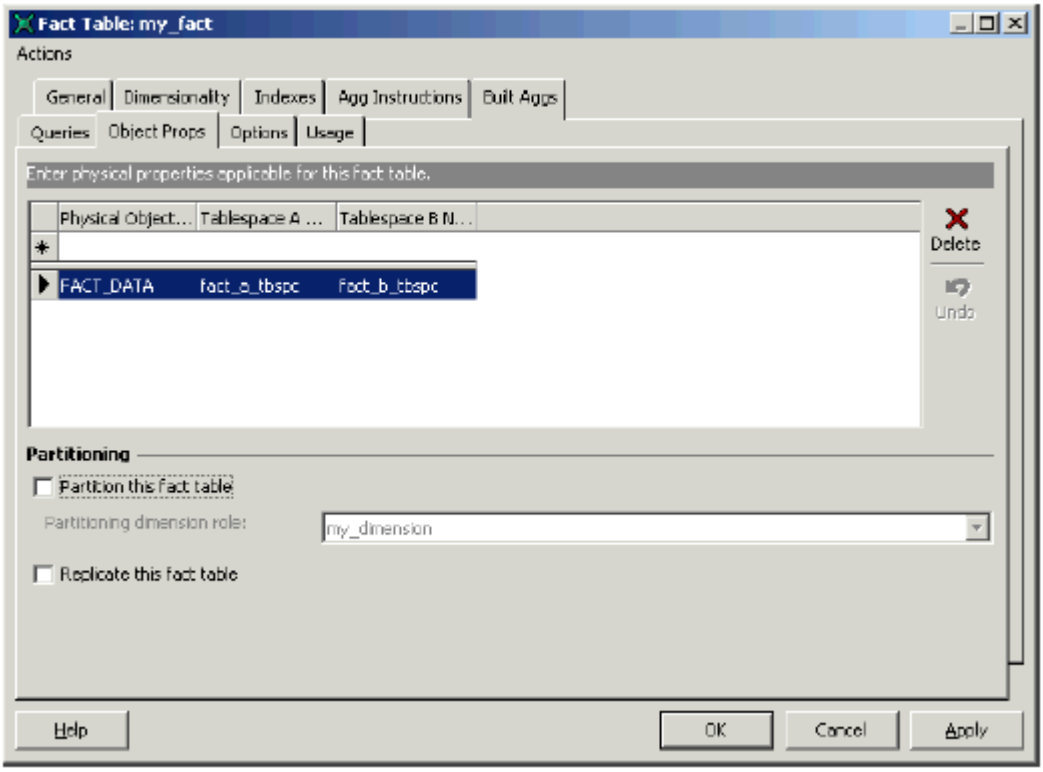


Figure 8: Fact Table Dialog Box

You can choose to partition or replicate the selected fact table by selecting the appropriate option. If you choose to partition your fact table, you must also select the partitioning dimension role. Partitioning in DB2 is done through hashing, but not range partitioning. See the DB2 documentation for more information.

For replication and partition considerations see "Partitioning DB2 Base Dimensions" on page 37.

Improving Query Performance

Infor Campaign Management supports several different methods that dramatically improve query performance, including aggregation, indexing, sampling, and table thinning. Each method accelerates different classes of queries.

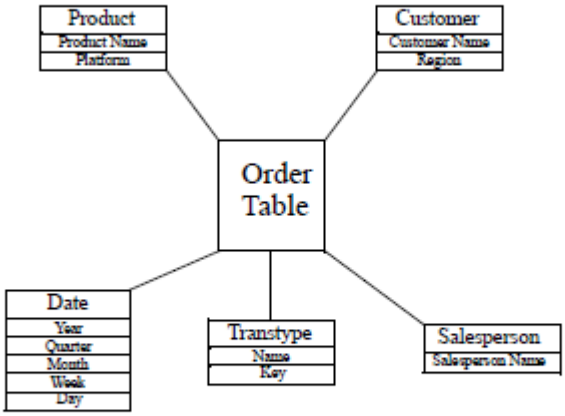
In general, the presence of aggregate tables accelerates queries that answer high-level business questions, such as Sales by Business Unit and Fiscal Year. Deep drill-down queries, or highly filtered queries that ask for a small subset of data, are typically enhanced by the presence of an index.

Aggregate Building

Creating aggregates speeds up system performance at the front end of Infor Campaign Management. This is because some queries simply require summary data, which can be pre-computed by the system rather than being recomputed for each query. For example, assume that a site has 5,000 products categorized by ten product types. Users may request reports that show data aggregated by product type. If there is a grouping by product type, the Infor Campaign Management aggregation program creates a table with ten rows (one for each product type) and pre-calculates “rolled-up” (combined) fact aggregates based on this smaller dimension table. By using significantly smaller fact aggregates, the system can generate reports considerably faster.

For example, your star schema can look like that in "Figure 9: A Star Schema" on page 43.

Figure 3-8: A Star Schema



An aggregate on this star might look like that shown in Figure 3-9.

Figure 9: A Star Schema

An aggregate on this star can look like that shown in "Figure 10: An Aggregate Star" on page 44.

Figure 3-9: An Aggregate Star

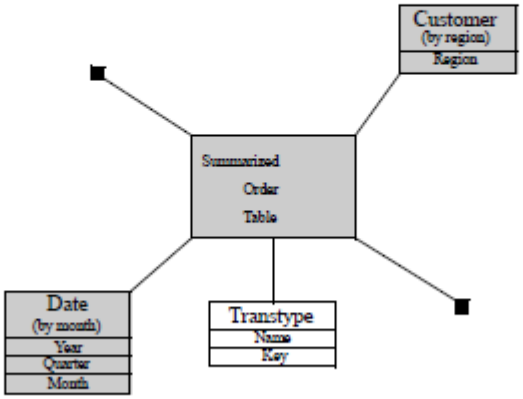


Figure 10: An Aggregate Star

This aggregate star is built using two dimension aggregates (for **Customer** and **Date**) and one complete dimension (**Transtype**). Two dimensions (**Salesperson** and **Product**) are not used at all in the aggregate. The aggregate fact table contains summarized information based on the included dimensions and dimension aggregates.

Note: Degenerate dimensions (dimensions that contain a single attribute column only) cannot be included in aggregates.

When the EpiMart database is successfully populated, the rows in the fact tables contain foreign keys to dimension table rows. In the EpiCenter, dimension primary keys are usually stored as integers. For example, fact rows may resemble those in fact table `Order_0` ("Order_0 Fact Table" on page 44), and dimension rows may resemble those shown in `Customer_0` ("Customer_0 Dimension Table" on page 44), `Product_0` ("Product_0 Dimension Table" on page 45), and `Salesperson_0` ("Salesperson_0 Dimension Table" on page 45).

Table 6: Order_0 Fact Table

Customer_key	Product_key	Salesperson_key	Total Sale
6	8	5	$20
5	6	6	$30

Table 7: Customer_0 Dimension Table

Customer_key	Customer Name	Region
5	ABC Company	West
6	Bill’s Lighting	West

Customer_key	Customer Name	Region
7	Fred’s General Store	East

Table 8: Product_0 Dimension Table

Product_key	Product Name	Platform
6	Product A1	Windows
7	Super X	Macintosh
8	Smash-Em	Linux

Table 9: Salesperson_0 Dimension Table

Salesperson _key	Salesperson Name
5	Gene
6	Sue

All fact and dimension tables that end with `_0` (underscore zero) are called base tables in the EpiMart database. They contain raw data at the level that is extracted from the source system.

A base dimension is a physical dimension table in the EpiMart database, such as **Customer** or **Product** . Base dimension tables contain the actual dimensional values that are available for reporting or filtering. Attributes, such as Customer Region or Product Platform, can be rolled up.

For example, end users often want to create reports of total sales by region. Suppose that `Order_0` has hundreds of millions of records, that **Customer_0** has many millions of records, and that there are only three different regions in the **Region** field. A report of sales by region against `Order_0` and **Customer_0** needs to aggregate over billions of records, causing the report to return answers very slowly. However, a pre-built aggregate with three (rolled-up) rows that already has the total sales by regions can answer the same question very quickly.

The names of aggregate tables end with _ n , where n is a number greater than zero. The aggregate tables for the `Order_0` fact table and `Order_0` dimension table can include the tables `Order_17` ("Order_17" on page 45) and `Order_0` ("Customer_1" on page 45).

Table 10: Order_17

Customer_key	Total_Sale
5	$50

Table 11: Customer_1

Customer_key	Region
5	West
7	East

Note the following:

- The `Order_17` table contains the same total sales dollar amount, but the number of rows is reduced (in a larger example, this reduction can be extremely significant). The name of the table is the same as the base table, with a different number at the end.
- The `Total_Sale` column in the `Order_17` table is the same data type as its corresponding column in the base `Order` table. Use data types that can store both individual fact values as well as aggregated facts over an entire table.
- The `Customer_1` table contains only the rolled-up field(s). The number of rows is reduced so that each row contains a distinct Region. The table name is the same as the base table, with a different number at the end.
- The foreign key `Customer_key` has the same name as in the base tables above, but the numbers differ. It is important to recognize that the `Customer_key` field in `Order_17` can be joined only with the `Customer_1` table, since `Order_17` is at the granularity of Regions, not Customers.

See "Base Dimension Tables" on page 100, for a description of how tables such as `Customer_1` are built using Admin Manager.

Aggregate Groups

Aggregate groups allow you to generate a collection of related aggregate instructions. For each dimension role that you include in the aggregate group, you specify the ways in which that dimension role can be included in a generated aggregate instruction. You can include a dimension role in its entirety, exclude a dimension role, or include a dimension role in aggregated form. To include a dimension role in aggregated form, you first need to define an aggregate on the underlying base dimension (see "Base Dimension Tables" on page 100, for details).

The fact aggregates that can be generated from an aggregate group are determined by a combinatorial expansion based on the dimension role possibilities that you set up.

For example, you can have the following specifications:

- The Product dimension can be included either as an aggregate named `product_type` or as an aggregate named `product_platform`.
- The Customer dimension can be included in its entirety, excluded, or included as an aggregate named `age_and_income`.
- The Household dimension can only be included as an aggregate named `region`.

Since there are two possibilities for the **Product** dimension, three possibilities for the **Customer** dimension, and one possibility for the **Household** dimension, this aggregate group can be used to generate the six aggregate instructions shown in "Table 12: Sample Generated Aggregates" on page 46:

Table 12: Sample Generated Aggregates

Aggregate	Product	Customer	Household
1	product_type	entire dimension	region
2	product_type	not included	region

Aggregate	Product	Customer	Household
3	product_type	age_and_income	region
4	product_platform	entire dimension	region
5	product_platform	not included	region
6	product_platform	age_and_income	region

The Aggregate Building Process

Note: All actions occur in the set of mirrored tables (A or B) that is not Current. See "Data Mart Mirroring: A and B Tables" on page 191 for more information.

When AggBuilder is first executed during an extraction job, the following actions are taken.

1. Dimension aggregates are built. If a base dimension column set is included in some aggregate instruction, then a dimension aggregate is built with that column set. (See "Defining Fact Aggregates" on page 128 for details of how to define aggregate instructions.) Indexes are built for these tables as appropriate.
2. For each fact table, AggBuilder first creates a unique table name and then builds and indexes a fact aggregate for each aggregate instruction. Indexes are built based on the definitions in the **Index** tab of the Fact dialog box.
3. AggBuilder records what it has built in a set of read-only metadata tables in the EpiMeta database. You can use Admin Manager to browse these lists. The Infor Campaign Management Server uses these same lists at runtime to decide which aggregates to use to satisfy an end-user query.
4. All activity is logged to the EpiCenter's logging tables.

   If aggregates have already been built for a fact or dimension table, then AggBuilder updates those aggregates, rather than rebuilding them, wherever possible.

Aggregate Optimization

Although the use of aggregates in an EpiMart significantly speeds up user queries, aggregates increase extraction time and consume disk space. A proliferation of aggregates greatly increases extraction time, and aggregates must be updated when new data is extracted. To ensure that extraction completes within a reasonable time, you must select aggregates carefully.

The best way to choose aggregates is based on actual usage. If your EpiMart and EpiOp databases reside on an Oracle or SQL Server database server, you can use the **Built Aggs** tab of the Fact or Dimension dialog boxes in Admin Manager to examine query statistics. Query statistics allow you to determine how well the currently defined aggregates match actual usage patterns. You can then construct an optimized set of aggregates based on user query logs.

You can also use the Reflection topic to analyze query logs. See chapter 12, "Configuring the Reflection Application," in the *Topic Implementation Guide* for details.

Dimension Query Statistics

Query logs allow you to examine the frequency with which combinations of dimension columns are queried. A user query typically involves retrieving fact quantities based on the associated dimension elements. The associated dimension elements are selected based on the values in dimension columns. Thus, a typical query selects fact quantities based on some subset of the columns in each dimension.

If some set of columns of a base dimension are used in a user query, then statistics about that column set can be viewed in Admin Manager. These statistics include information about how often that column set is queried and whether a dimension aggregate can be used for such a query.

You can use these statistics to fine-tune your dimension aggregate definitions. If a set of columns are queried often then, ensure that a closely matching aggregate is available to speed up these queries. On the other hand, if you find that an aggregate is infrequently used by actual queries, you can improve extraction time without significantly slowing query response time by removing that aggregate.

Fact Query Statistics

Admin Manager also allows you to examine fact query statistics. Every fact table query is based on the values of some set of dimension roles, and those dimension role values are found based on the values in some set of columns of the associated base dimension. Thus, every fact table query has associated with it several sets of dimension columns, one for each dimension role used.

Given a fact table and an associated collection of dimension roles and dimension column sets, if the fact is queried based on that collection, then query statistics about that fact table and the associated collection can be viewed in Admin Manager.

These statistics include information about how often the fact table is queried with that collection of dimension roles and dimension column sets, and whether a fact aggregate can be used for such a query. As with dimension aggregate statistics, you can use these statistics to fine-tune your fact aggregate definitions.

Fact Indexing

A fact base table is usually the largest table in EpiMart. Even tables with millions of rows can be accessed quickly via an index if only a small number of data pages are needed to service a query.

A typical Infor Campaign Management query uses a single fact table (either base or aggregate). As a user drills down into a result set, the physical table that services each query generally moves from a high-level aggregate (with few rows) to a larger aggregate (possibly using an index on the aggregate). For the user's response time to be acceptable in each of these cases, you must properly configure both aggregates (as described in the previous section) and indexes.

All aggregates are indexed in the same manner as the base table (insofar as the aggregate has the same dimensionality as the index). For example, if the base table has a composite index on the ordered set of columns `customer_key` , `product_key` , and `territory_key`, then an aggregate that contains only the Customer and Territory dimensions has a composite index on `customer_key` and

territory_key. If two different indexes on the base table result in the same index on the aggregate, then the aggregate index is built only once.

By default, each fact table has one index built on its date dimension key during extraction. An index is built by default on date_key because a wide class of queries filters on the time dimension. For example, if a system contains data for multiple years and a user asks for a single calendar month, the database almost certainly use an index when it services this query (whether via the base table or an aggregate).

An index is the metadata definition of indexes to build on fact tables in the EpiMart database. You can use the **Indexes** tab in the Fact Table dialog box in Admin Manager to do the following:

- Configure indexes on a fact table
- Select the dimension roles index and order contains

It often makes sense to build several different indexes, each with a different dimension role as the leading term. The leading term of an index is the most important term. For example, a telephone book is indexed by last name and then first name. You cannot locate someone by first name only, but you can use the first name to narrow your search after you have located the last name.

If, for instance, end users often filter on attributes of the Customer and Product dimensions, then you can build an index on the combination (customer_key , product_key) and another on product_key by itself. There is no need for the second index to include customer_key, because joint customer/product queries can already be answered by the first index.

Data Sampling

Sampled EpiMart tables contain statistical samples of the data in the main EpiMart tables. Sampled EpiMart tables are used whenever an end user requests fast approximate counts. The base tables are used for other list and campaign operations, including exact counts and list display and download.

Sample tables are based on statistical samples of all dimensions that have the **Enable sampling for this dimension** option checked in the **Options** tab of the Base Dimension dialog box. You can specify the sampling size in this dialog box as well.

Note: Be sure to enable sampling only when the sample size is statistically accurate. For example, if you have an Organization dimension that contains only 10 organizations, sampling this dimension can lead to inaccurate approximate counts.

Relations

Modeling naturally occurring dependencies or relationships between dimensions is one of the most powerful features in Infor Campaign Management. Such relationships make it possible to construct complex queries that span dimensions. The relationship between two dimensions is modeled in an EpiCenter by a relation.

A relation defines a relationship between two list-producing base dimensions (a base-dimension is list-producing if it is defined with the **Lists can be made from this dimension** option selected).

You can define a relation between any two dimensions in your EpiCenter, and these relations can be one-to-one, one-to-many, or many-to-many. You can also design a rich, multi-level hierarchy of relations, such as the sample hierarchy displayed in "Figure 11: Sample Relation Hierarchy" on page 50.

Figure 3-10: Sample Relation Hierarchy

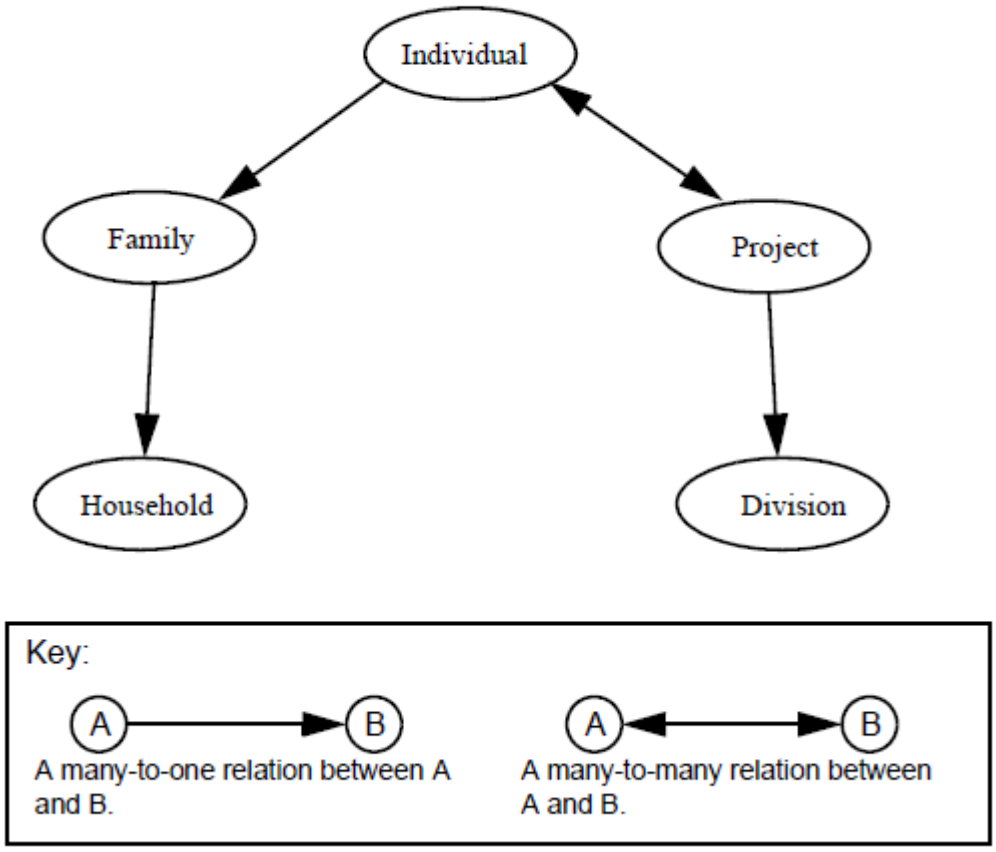


Figure 11: Sample Relation Hierarchy

In this example, the Individual dimension has a many-to-one relationship with the Family dimension, which has a many-to-one relationship with the Household dimension. Individuals and Projects share a many-to-many relationship, and the Project dimension has a many-to-one relationship with the Division dimension.

When querying a dimension, users can access the attributes of all dimensions that have a provable relationship with the dimension. For example, because Individual has a many-to-one relationship with Family, and Family has a many-to-one relationship with Household, Individual has a many-to-one relationship with Household. Therefore, when making a query on the Individual dimension, users can use attributes from the Household dimension to limit their query results.

Relation Fact Tables

A relation fact table is a special fact table that defines a relationship between two dimensions. For every pair of dimension values, the relation fact table contains a row with a `member` column value of 1 if the two values are related, and no row otherwise.

Infor Campaign Management determines whether two values are related based on the sum of the `member` column for all relation fact table rows that relate the two values. If this sum is greater than 0, the two values are related.

If you wish to change a relationship between two dimension values, the pair must be debooked in the fact table (by setting the value of `member` to -1). For example, Table "Relation Fact Tables" on page 51 shows two rows from a relation fact table that relates the Individual and Team dimensions.

Table 13: Sample Relation Fact Table

Individual	Team	Member
Adam Vinatieri	New England Patriots	1
Jeff Garcia	San Francisco 49ers	1

If Adam Vinatieri is traded to the San Francisco 49ers, a debooking row must be added to the table to remove his relationship with the New England Patriots (Member = -1), as illustrated in the table below "Relation Fact Tables" on page 51. The last row makes him a member of the San Francisco 49ers (Member = 1).

Table 14: Debooking in Sample Relation Fact Table

Individual	Team	Member
Adam Vinatieri	New England Patriots	1
Jeff Garcia	San Francisco 49ers	1
Adam Vinatieri	New England Patriots	-1
Adam Vinatieri	San Francisco 49ers	1

You use semantics to transform data from your source systems into the format described above. Infor recommends using Statelike semantics for relation fact tables. See "Semantics," on page 223 for information.

Explicit and Implicit Relations

Relations can be defined either explicitly , with a relation fact table, or implicitly , by the way that other explicit relations have been defined in a hierarchical relationship. For example, if there is an explicit relation between families and individuals, and another explicit relation between households and families, there exists an implicit relation between households and individuals.

Note: You cannot create an explicit relation between two dimensions that are already related implicitly. Therefore it is illegal to create a relation fact table for any relation that is defined implicitly by other relations.

All explicit relations are preprocessed by Infor Campaign Management into a consolidated relation table that facilitates fast query performance. You can also preprocess implicit relations into a consolidated relation table if you expect the relationship to be traversed frequently in end-user queries. To indicate that an implicit relation must be preprocessed, define it in Admin Manager. Any implicit relation that is not defined in Admin Manager can still be traversed by end-user queries, but runtime performance is significantly lower.

Implicit Many-to-Many Relations

A relation between dimension A and B is outwardly-branching from A to B if more than one B can be assigned to a single A. Many-to-many relations are outwardly-branching both when moving from A to B and when moving from B to A, while a one-to-many relation is only outwardly-branching in one direction, and a one-to-one relation is outwardly-branching in neither direction.

Due to the large result size of implicit many-to-many relations, Infor Campaign Management only supports implicit relations between dimensions that are connected in at least one direction with no more than one outwardly-branching relationship.

"Figure 12: Explicit and Implicit Relation Hierarchy" on page 53 illustrates the legal implicit relations that can be deduced from the original sample hierarchy:

- The implicit relationship between Individual and Household is one-to-many because there are zero outwardly-branching relations on the path from Household to Individual, and two outwardly-branching relations from Individual to Household. This relation is legal because there is no more than one outwardly-branching relation in the direction from Household to Individual.
- The implicit relationship between Project and Household is many-to-many because there is one outwardly-branching relation in the direction from Household to Project (between Individual and Project) and three outwardly-branching relations in the direction from Project to Household. This relation is legal because there is no more than one outwardly-branching relation in the direction from Household to Project.
- Other implicit many-to-many relations include Project to Family, and Division to Individual.

Figure 3-11: Explicit and Implicit Relation Hierarchy

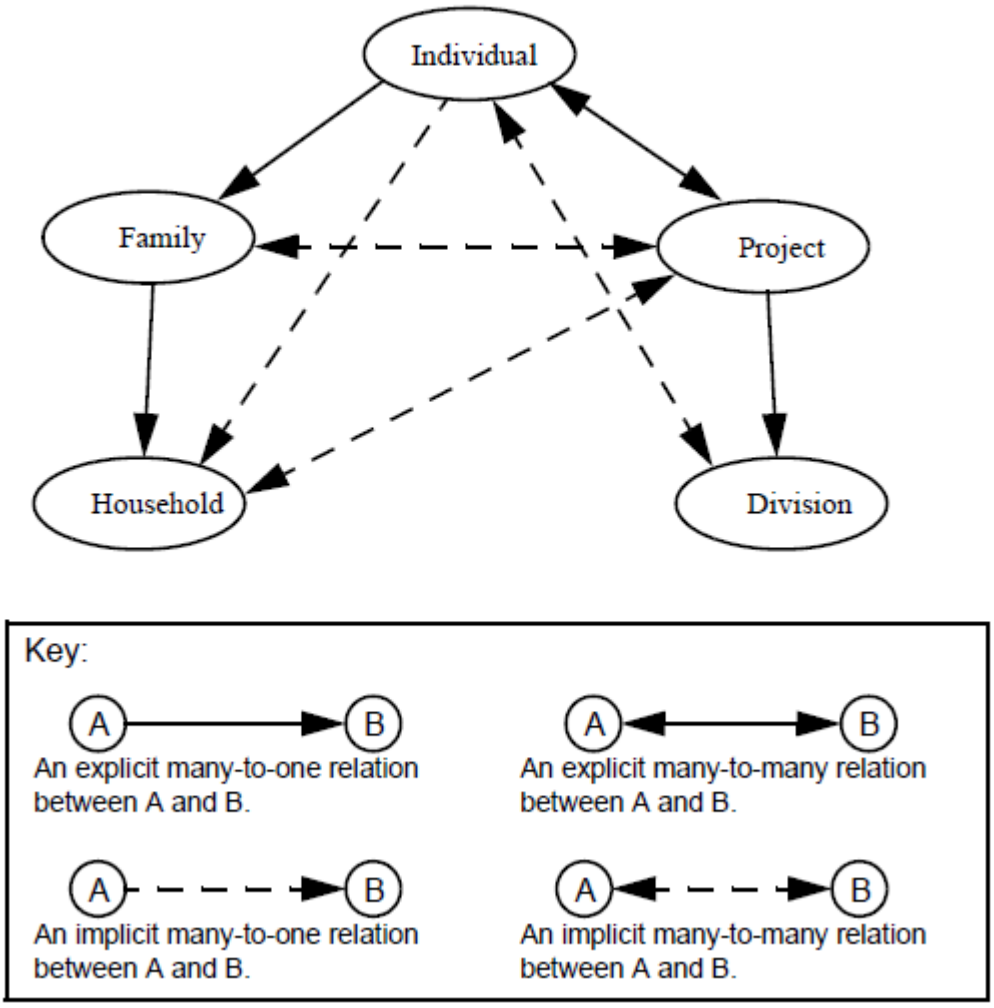


Figure 12: Explicit and Implicit Relation Hierarchy

- The implicit relationship between Division and Household is illegal because there are two outwardly-branching relations in the direction from Household to Division, and three outwardly-branching relations in the direction from Division to Household. Since at least one direction must have one or fewer outwardly-branching relations, the relation is illegal.
- Note that the implicit relationship between Division and Family is also illegal.

  Note: Unique Index Errors generated during the Relations phase of MomBuilder occur when a one-to-one or one-to-many relationship is violated in a relation fact table.

Star-Schema Extensions for Infor Campaign Management

The Infor Campaign Management application allows Infor Campaign Management users to create lists and campaigns. A campaign refers to a single marketing program. For example, when you send one thousand pieces of targeted mail, you have run a campaign. A campaign is comprised of a list of campaign recipients that is subdivided into multiple segments, called cells . Each cell is assigned a particular promotion (or communication) that can optionally include one or more targeted messages .

A message is any element of a treatment that can or can not be sent to a campaign recipient. For example, a campaign can include an email newsletter (the treatment) that contains multiple articles (the messages), each targeted for the individual recipient. One recipient can receive the newsletter with messages A and B, while another recipient can receive the newsletter with messages B and C. You can also create treatments that do not use messages at all. In that case, all recipients of the treatment receive the same content.

Infor Campaign Management uses special tables and indexes that are collectively known as campaign accelerators . The MomentumBuilder utility maintains the campaign accelerators (see "MomBuild" on page 299), and must run every time you perform an extraction.

The following sections describe the built-in campaign tables and accelerators. See the *Topic Implementation Guide* for additional details about configuring Infor Campaign Management.

Backfeed Tables

Every time a campaign is run, information about that campaign is recorded in backfeed tables stored in the EpiOp database. These tables include information about the cells, treatments and messages that are included in the campaign. Backfeed tables serve as source tables for information that can be later extracted into the built-in campaign dimension and fact tables described below. The existence of the backfeed tables allows users to analyze campaigns that they have run in the past, as well as exclude customers from receiving campaign treatments after having received other treatments in the recent past.

For more information on backfeed tables, see "Backfeed Merging and Extraction" on page 173, and "Mirroring in Backfeed Tables" on page 191.

Built-in Base Dimension Tables

An Infor Campaign Management application that makes use of campaigns requires the following built-in base dimension tables:

- **Campaign**
- **Cell**
- **Message**

An EpiCenter uses these base dimensions to report on the campaigns that the Infor Campaign Management system generates. Reported items include campaign name, expected revenue, and type of message. See "Appendix D, “Campaign-Related Dimensions,”" on page 509 for a complete description of the columns of the **Campaign** , **Cell**, and **Message** dimensions.

If you need to record additional campaign information in the Campaign, Cell , and Message dimensions, you can define your own columns. End users can specify values for these columns if you define them with VARCHAR, FLOAT, DATE or INT data types.

Built-in Fact Tables

Each dimension that can be targeted by a campaign requires a set of campaign-related fact tables. These tables are automatically generated by Infor Campaign Management. The following sections describe these fact tables.

The Communication Fact Table

When a base dimension is marked as list-producing and backfed (specified in Admin Manager in the **Options** tab of the Base Dimension dialog box), a Communication fact table is automatically generated for the dimension. The fact table is named dimensionName_shortID _com, where dimensionName is the first 13 characters of the base dimension name, and shortID is the 2-character abbreviation for the dimension (as specified in the Base Dimension dialog box).

The Communication table is designed to hold information about treatments that have been applied to campaign recipients. Ordinarily, data for this table is extracted from the Communication Backfeed table for the dimension, which is automatically populated by the Infor Campaign Management Server.

The Communication fact table contains one row for each member of the list-producing dimension that is targeted by a campaign. This fact table captures information related to the communication history of all campaigns (current and previous marketing programs) and makes it available to the data mart for query. The Communication fact table includes foreign keys to the Date, Transtype, Campaign, and Cell dimensions, as well as a key to the list-producing dimension for which it is named.

Note: Communication tables do not store information about seed records. See "The Seed Fact Table" on page 56 for a description of seed records.

The Message Fact Table

The Message table is designed to hold information about the individual messages that have been sent to campaign recipients. Ordinarily, data for this table is extracted from the Message backfeed table for the dimension, which is automatically populated by the Infor Campaign Management Server.

When a base dimension is marked as list-producing and backfed (specified in the **Options** tab of the Base Dimension dialog box), a Message fact table is automatically generated for the dimension. The fact table is named dimensionName_shortID _msg, where dimensionName is the first 13 characters of the base dimension name, and shortID is the 2-character abbreviation for the dimension.

The Message fact table is designed to contain one row for each message that a member of the list-producing dimension receives in a campaign (making it larger than its corresponding Communication fact table). The Message fact table includes foreign keys to the Date, Transtype, Campaign, Cell, and Message dimensions, as well as a key to the list-producing dimension for which it is named.

The Inferred Response Fact Table

The `InferredResponse` table, which has a structure similar to that of the `Communication` table, is designed to hold information about inferred responses to campaigns that have been run. Sometimes you do not have direct access to data about campaign responses, but information about such responses can be inferred from other available data.

For example, if you send a communication to a particular customer and the customer buys a product from you within 90 days, you can infer that the communication was successful. In such a situation, an end user can run a special kind of campaign (an inference campaign) that is designed to create such inferences. The data from an inference campaign can then be extracted from the Communication Backfeed table into the `InferredResponse` fact table for the dimension.

When a base dimension is marked as list-producing and available for inferred responses (specified in the **Options** tab of the Base Dimension dialog box), an Inferred Response fact table is automatically generated for the dimension. The fact table is named dimensionName_shortID _ifr, where dimensionName is the first 13 characters of the base dimension name, and shortID is the 2-character abbreviation for the dimension.

The Seed Fact Table

Seeds are dummy records that are included in campaigns that target members of the dimension. They can be used to verify that campaigns have been executed, or that fulfillment houses have not sold your campaign information to other vendors. For example, an Admin Manager can include an address to which she has access so that she receives the communications that are included in her campaigns. This address acts as a seed record in the campaign, and is not used in campaign analysis.

Setting a base dimension as list-producing and seed-using—specified in the **Options** tab of the Base Dimension dialog box—automatically generates an empty seed fact table for that dimension. The fact table is named dimensionName_shortID _sed, where dimensionName is the first 13 characters of the base dimension name, and shortID is the 2-character abbreviation for the dimension.

Seed records are not included in the following:

- list/campaign counts
- communication (backfeed) tables

See "Fact Tables" on page 120 for details on how to set up seed tables in the EpiMart.

Planning Your EpiCenter

Before you configure an EpiCenter, you need to analyze your site's business model to determine the following:

- How many EpiCenters are appropriate?

  An EpiCenter is a construct for organizing your metadata into a database called EpiMeta, which defines the structure of an EpiMart database. The EpiMart database is where the data extracted from your source systems resides. If your system has adequate memory and storage (see the *Infor Campaign Management Installation Guide* for details of system requirements) and your business model warrants it, you can set up multiple EpiCenters. For example, you can have one EpiCenter for development and one EpiCenter for production.

- What dimension and fact tables are required for the EpiCenter?

  The attributes and measures that end users can wish to query determine which dimension and fact tables you configure. These tables, which are global to the EpiCenter, must relate to the topics that you configure, since the web pages at the nodes of a topic can use the data in these tables.

- What dimension roles are associated with dimension tables?

  Each base dimension table in your EpiCenter must have at least one associated dimension role in order to be available to the data mart. When a physical base dimension table serves multiple purposes, such as a Customer base dimension table that provides Ship-to and Bill-to data, you can need to define a dimension role for each usage.

- Which dimensions are degenerate dimensions?

  A degenerate dimension can be thought of as a dimension with only a single attribute. Degenerate-dimension values are stored directly in the fact table, rather than in a separate dimension table that is referenced by a foreign key in the fact table.

- What is the extraction process for your site?

  Analyze the kinds of jobs and job steps needed for extraction of source data. The available job steps are described in "Database Extraction Overview." on page 169 You can want to create job flow charts to help you with this process. After you design the extraction process at this higher level, you must be better able to do the following:

  - Identify the extraction steps (extraction commands and semantic instances) needed to extract information from your source systems and write these steps
  - Define external tables
  - Identify requisite system-call commands

Before you can begin to implement the schema for your EpiCenter, you must identify potential queries and the source data for those queries. You must then map the columns from the source-system tables into the star schema for your data mart, and select the semantics that govern how new data rows are to be incorporated into these tables.

Consider which Infor Campaign Management applications your user base is required. Then consider the source data that those users need to query and the requisite level of granularity of that data. Finally, determine which attributes and measures provide appropriate data access for users. When you have identified the fact and dimension columns, the schema that you design allows you to create a global pool of measures and attributes that users can combine in any meaningful way.

Introduction to Admin Manager

4

This chapter introduces Admin Manager, the graphical utility that you use to configure your EpiCenter and Infor Campaign Management applications. By default, Admin Manager is installed on the host that runs the Infor Campaign Management Server. You can also install copies on other Windows hosts using the Custom setting of the Infor Campaign Management installation program.

To open Admin Manager from the **Start** menu, choose **Start** , then:

- **Program > 10.1.0 > Admin Manager > Admin Manager**

The sections in this chapter describes various features that appear throughout Admin Manager's user interface. Detailed information about specific dialog boxes is provided in an online help system and in later chapters. See "Infor Campaign Management Utilities" on page 381 for information about the command-line interface and other utilities related to Admin Manager.

Note: When connecting to the EpiMeta database over a high-latency wide-area network (WAN), Admin Manager is optimized for use with a Terminal Server (Remote Desktop) connection to a host that is running Admin Manager and is on the same local area network (LAN) as the database server.

Note: If you have installed the 64 bit Campaign Management application, you cannot directly access Admin Manager from Start -> Programs or by directly double clicking on ' `epimgr.exe` ' under <OM_HOME>\bin. To access Admin Manager, you need to open a command prompt (with administrative access), navigate to the <OM_HOME>\bin folder, and execute these commands:

```
Set JAVA_HOME=<32 bit JDK/JRE location>

Set PATH=%JAVA_HOME%\bin;%PATH%

Epimgr.exe
```

Admin Manager Window

The main Admin Manager window ("Figure 13: Admin Manager Window" on page 60) displays a hierarchy of folders and object types in the left display pane. Each object type represents a dialog box

that you use to edit an element of your EpiCenter metadata. Each folder contains multiple object types and represents a stage in the process of configuring an EpiCenter and application.

Figure 4-1: Infor Marketing Manager Window

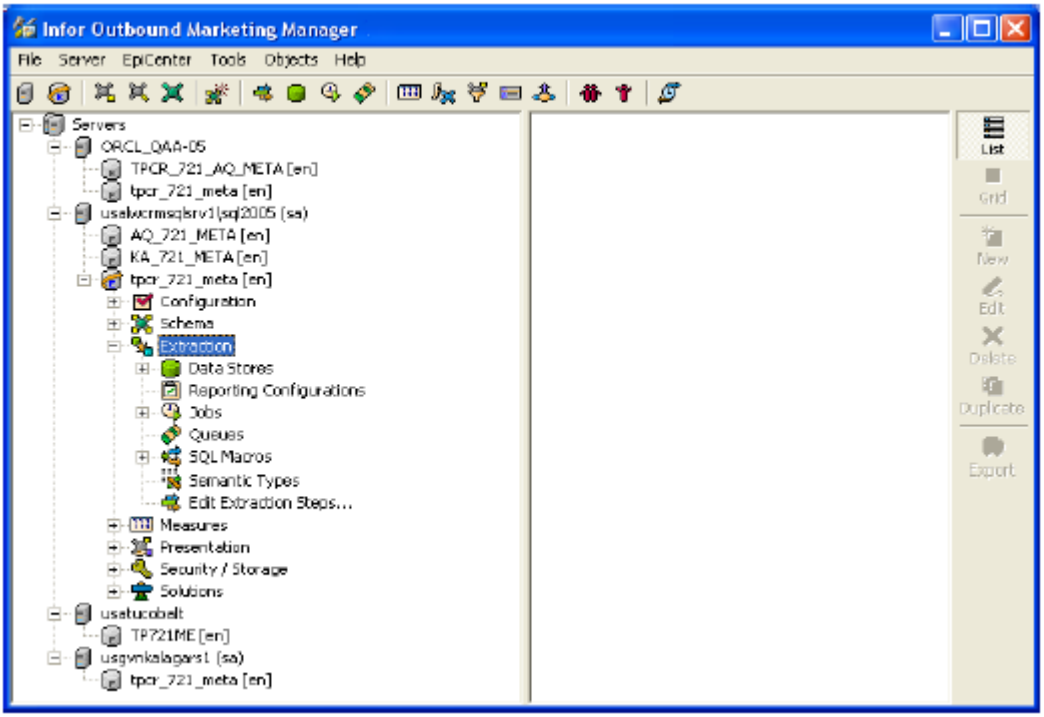


Figure 13: Admin Manager Window

When you double-click the icon for a folder, Admin Manager alternately displays or hides the object types within that folder. When you double-click an object type, Admin Manager displays the dialog box for a new instance of that object. When you select an object type in the left pane, saved instances of that object are listed in the right pane.

You can right-click any object in the main window to display a pop-up menu of actions that are associated with that object.

In addition to the display pane, Admin Manager's window also includes a menu bar, and two toolbars. The menu bar provides access to commands that you use to initialize and maintain applications, such as registering database servers and EpiCenters (data marts), generating the schema for a data mart, performing consistency checks, importing and exporting metadata, and so on.

The top tool bar displays buttons that allow you to configure new data mart and application elements and invoke specific actions that relate to your entire EpiCenter. This toolbar is shown in "Figure 21: Admin Manager Toolbar" on page 72. The side toolbar allows you to invoke specific actions that relate to a selected object type.

The Grid Editor

You can edit individual objects either by opening the dialog box for that object, or by using the grid editor. The grid editor is available by right-clicking an object type and selecting **Edit <object> with grid** , or by clicking the **Grid** in the side toolbar of the main window.

To edit an object with the grid editor, click in any field of that entry and then enter your updates. Some fields behave like text boxes and accept edits in place. Others display drop-down menus from which you choose from among preset values.

No edits made within the main form grid editor are committed to the database until you disable the grid by selecting another object type, or by clicking the **List** button in the side toolbar.

Note: If you click **Grid** in the main form and then edit one of the objects in the grid by opening the object's dialog box, the grid is disabled when you return to the main form.

Sub-Folders

You can organize the elements of the same type by using sub-folders. Only items that have been placed within a folder are displayed in the right pane when the folder is selected.

To create a subfolder, right-click any object type and select **New Folder**. The folder is displayed under the selected object type in the EpiCenter tree hierarchy. (See "Figure 14: Nested Sub-Folders" on page 61.) Items can be drag-and-dropped from the right pane into folders of the same object type. Other folders can also be drag-and-dropped into a folder to create an arbitrarily nested hierarchy.

Figure 4-2: Nested Sub-Folders

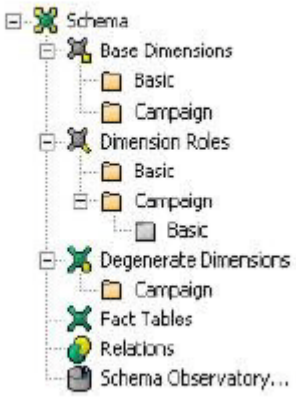


Figure 14: Nested Sub-Folders

Built-in Objects

All initialized EpiCenters contain built-in objects that are provided by Infor. These objects include the Date and Campaign dimensions, default extraction jobs, attributes, topics, and web pages. Built-in objects cannot be deleted and are kept in separate sub-folders from other objects that you configure. You can modify a built-in object by copying it and then making changes to the copy.

Top-Level Folders

Top-level folders that appear in Admin Manager window include:

- Configuration
- Schema
- Extraction
- Measures
- Presentation
- Security/Storage

Depending on how you have configured your Admin Manager preferences, the folders within an EpiCenter hierarchy can be displayed in a single tree or in a two-pane view like the display of Windows Explorer.

The sections that follow describe the folders that are discussed in this guide and in the *Infor Campaign Management Topic Implementation Guide*.

The Configuration Folder

"Figure 15: The Configuration Folder" on page 63 illustrates the icons that appear within the Configuration folder. The icons in this dialog box open tabs within the Configuration dialog box. These tabs allow you to specify global resources such as the local language, measure units for transactions, and so on.

For more information about this dialog box and the tabs that it contains, see "The Configuration Dialog Box" on page 147.

Figure 4-3: The Configuration Folder

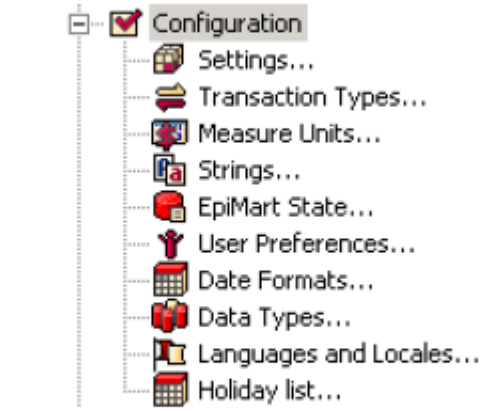
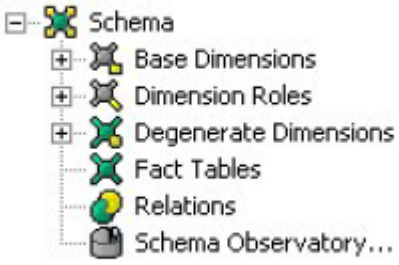


Figure 15: The Configuration Folder

The Schema Folder



The figure above illustrates the icons that appear within the Schema folder. These icons open various dialog boxes that allow you to configure the dimensions, facts, and other essential aspects of your data mart schema. See "Defining the EpiCenter Schema" on page 85 for information about how to implement your data mart schema.

The Extraction Folder

"Figure 16: The Extraction Folder" on page 64 illustrates the icons that appear within the Extraction folder. These icons open dialog boxes that allow you to configure:

- The extraction steps, including SQL statements and semantic instances, that are used in extraction jobs.
- The extraction jobs and job steps that EpiChannel runs to perform each extraction job.
- The macros that are expanded within extraction jobs to resolve table names and other values.
- The scheduling queues for extraction jobs and other batch operations.

See "Database Extraction Overview" on page 169 for information about extraction jobs.

Figure 4-4: The Extraction Folder

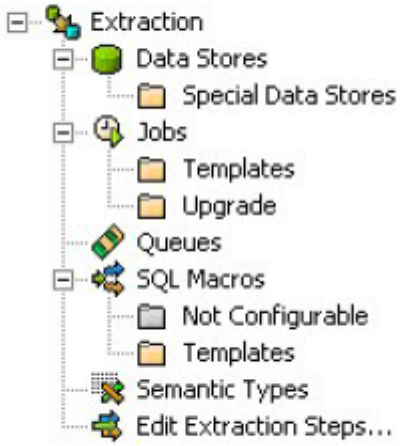


Figure 16: The Extraction Folder

The Measures Folder

"Figure 17: The Measures Folder" on page 65 illustrates the icons that appear in the Measures folder, which allow you to define measures and to configure presentation objects that allow your measures to appear within web pages.

See Chapter 5 of the *Infor Campaign Management Topic Implementation Guide* for information about measures.

Figure 4-5: The Measures Folder

Figure 17: The Measures Folder

The Presentation Folder

"Figure 18: The Presentation Folder" on page 66 illustrates the icons that appear in the Presentation folder. These icons open dialog boxes that allow you to:

- Define attributes.
- Configure the attribute layouts by which attributes are displayed in web pages.
- Create transaction filters for use when generating lists (see “Transaction Filters,” in the *Infor Campaign Management Topic Implementation Guide*).
- Create web pages.
- Create topics.
- Create link categories.
- Add entries to the glossary (see the section “Adding Glossary Entries” in the *Infor Campaign Management Topic Implementation Guide*).

Figure 4-6: The Presentation Folder

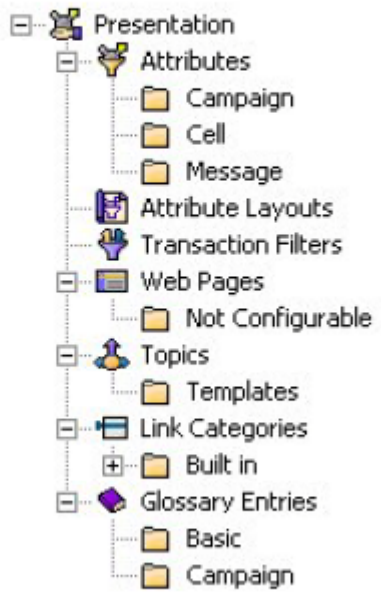


Figure 18: The Presentation Folder

The Security/Storage Folder

"Figure 19: The Security/Storage Folder" on page 67 illustrates the icons that appear in the Security/Storage folder, which allow you to:

- Add and remove groups and users.
- Define output processors for campaigns (see the section “Output Processors” in the *Infor Campaign Management Topic Implementation Guide*).
- Configure touchpoints used for campaign output formats.
- Add LDAP servers for use with lightweight-directory-access-protocol user authentication.
- Configure Foreign Systems for Marketing Resource Management integration.
- Configure the Report Gallery (see the section “The Report Gallery” in the *Infor Campaign Management Topic Implementation Guide*).
- View archived campaigns (see the section “The Campaign Archive” in the *Infor Campaign Management Topic Implementation Guide*).

Figure 4-7: The Security/Storage Folder

Figure 19: The Security/Storage Folder

Preferences

You can set display preferences for Admin Manager in the Preferences dialog box (see "Figure 20: The Preferences Dialog Box" on page 68). To open the Preferences dialog box, select **Preferences** from the **File** menu. The Preferences dialog box has four tabs: **General** , **Colors** , **Startup** , and **View** .

Figure 4-8: The Preferences Dialog Box

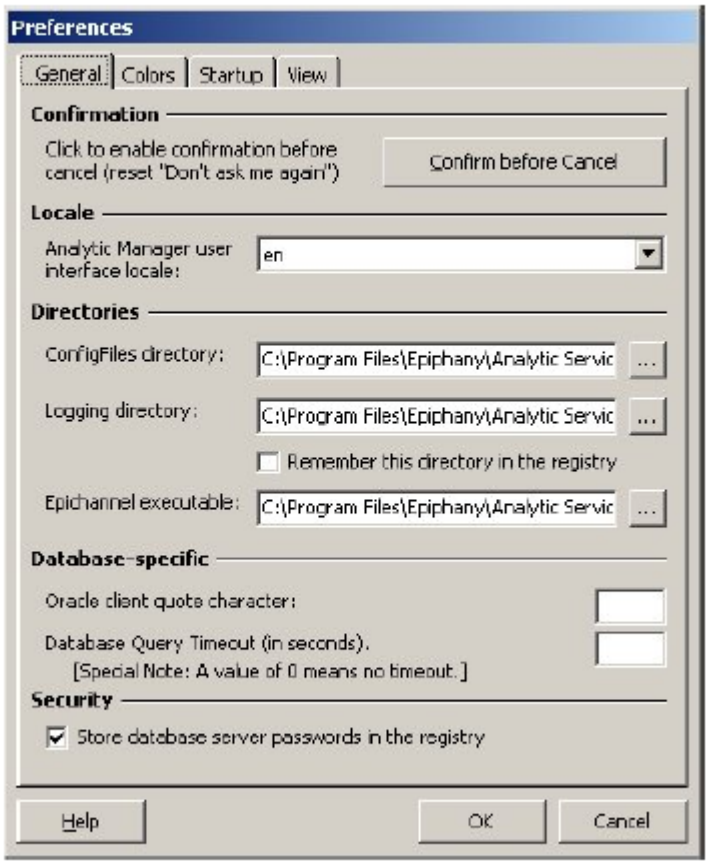


Figure 20: The Preferences Dialog Box

The General Tab

Following changes are available in the **General** tab:

- Confirmation

  Use **Confirm before cancel** to reset the confirmation option in dialog boxes. By default, you are reminded to save changes in every dialog box. You can turn this option off for individual dialog boxes by checking **Don't ask me this again** in the confirmation dialog box. To reset reminders for all dialog boxes, click **Confirm before Cancel** in the Preferences dialog box.

- Locale Admin Manager

user interface locale list to specify the language of Admin Manager's user interface if you have installed multiple languages. See the *Infor Campaign Management Installation Guide* for details about installing more than one language.

- Directories
  - The ConfigFiles directory text box allows you to specify the base directory for your Admin Manager configuration files, including all scripts. See "Admin Manager Scripting Interface" on page 396 for more information about scripts. about scripts.
  - The Logging Directory text box allows you to specify the base directory for all Infor Campaign Management logfiles. Select **Remember this log dir in the registry** if you wish to update the registry with this setting.
  - The EpiChannel Executable text box allows you to specify the directory in which the Epichannel executable is stored.

- Database-specific
  - See "Running Jobs with EpiChannel" on page 351 for information about EpiChannel.Se
  - The **Oracle Client Quote Character** parameter appears only if you use Oracle.

    Oracle 8x clients execute SQL statements properly if a single quote is used and fail if a double quote is used. Oracle 9x and later clients execute SQL properly if a double quote is used and fail if a single quote is used.

    Since Infor Campaign Management 10.0 does not support Oracle 8x clients, the default quote character must be left blank normally (to use the default) or set to a double quote. If quote characters are interpreted differently in your installation, you can change the quote character with this parameter. The change takes effect only for the current Admin Manager session.
  - Use the **Database Query Timeout** field to set the length of time Infor Campaign Management waits for a response from the database before timing out. If not set, Infor Campaign Management uses the default timeout as defined by the ODBC client, which is typically 30 seconds. Use this option with care, as setting the timeout too large can cause Infor Campaign Management to hang while waiting for a database operation to return. However, in some cases—for example, with a remote database or a complicated import on a small machine—a larger timeout is required due to the limitations of distance or database speed.

- Security

  Unselect the **Store database server passwords in the registry** option if you do not want Admin Manager to save the EpiMeta password in the registry. If it is not saved in the registry, Admin Manager prompts for it when users open a metadata database. If the password does not exist, or if Windows Authentication is in use, Admin Manager does not prompt for the password.

The Colors Tab

The **Colors** tab enables you to set colors for SQL-syntax highlighting within Admin Manager. This applies to SQL-related dialog boxes, such as the SQL Statement and SQL Macro dialog boxes.

Clicking an item in the list shows the current color in the **Sample** text box. To change it, click **Change**, and select a new color from the color menu, which updates the sample.

Click **Restore Defaults** to restore all colors to the default Infor Campaign Management settings.

The Startup Tab

The **Startup** tab allows you to specify which EpiCenters should be automatically opened when you start Admin Manager.

The View Tab

Following specifications are available in the **View** tab:

- The Toolbar Buttons area allows you to choose whether toolbar buttons in dialog boxes show as images only, or as images and text.

  Note: The Images and Text option for toolbar buttons is only supported at monitor resolutions of 1024x768 and above.

- The Navigator area allows you to switch between single pane and dual pane views.
- The Folders area allows you to specify whether you wish to enable the use of subfolders within your EpiCenter. Subfolders allow you to organize elements of the same type into separate areas.
- For example, you can store predefined attributes in a folder named `Built-in`, and attributes that you have created in another subfolder named `Custom`. Both folders appear under Attributes in the EpiCenter tree hierarchy.

Menus

Admin Manager provides the following menus that allow you to register and unregister database servers and EpiCenter data marts, set user preferences, invoke commands that update the schema of your data mart, and perform other maintenance tasks:

- **File**

  This menu contains the **Preferences**, **Instance Management** and **Exit** commands.

  - **Preferences**: This command displays the Preferences dialog box (see "Preferences" on page 67).
  - **Instance Management**: This command displays the Instance Management dialog box which you can use to add, update and delete Infor Campaign Management instances.
  - **Exit**: This command exits Admin Manager.

- **Server**

  This menu contains the **Register** and **Unregister** commands, which you can use to register or unregister a selected database server.

- **EpiCenter**

  This menu contains commands that apply to a specific data mart. These commands are also available by right-clicking the folder of a data mart.

- **Tools**

  This menu contains utility commands that allow you to modify or upgrade your data mart.

- **Object**

  This menu contains submenus related to metadata objects that appear in the main Admin Manager window.

- **Help**

  This menu allows you to browse the online help for Admin Manager either by topic or by index. It also allows you to display the About this Application dialog box for version-number and terms of use information.

The Main Toolbar

Many data mart operations can be performed using the toolbar located at the top of Admin Manager's window. "Figure 21: Admin Manager Toolbar" on page 72 shows this toolbar. The accompanying legend defines each icon.

Dialog-Box Features

Admin Manager dialog boxes include powerful features that simplify the process of configuring your data mart and applications. The sections that follow describe these features.

Object Names

Objects in Admin Manager are assigned three names: the name, the label, and the developer label.

- The **Name** text box specifies the object name that is used within the EpiMeta database.
- Object names can be written in any supported language unless the object is located within the Schema folder. Schema folder object names can only use ASCII characters.

Figure 4-9: Infor Marketing Manager Toolbar

Figure 21: Admin Manager Toolbar

	Register server		Register data mart		New dimension
	New dimension role		New fact		Generate schema
	Edit extraction steps		New data store		New job
	Newqueue		New measure		New fact term
	New attribute		New web page		New topic
	Newgroup		New user		Job Control

- The **Label** text box specifies the name that appears in the Infor Campaign Management end-user interface. The value that appears in this text box changes depending on the EpiMeta language configuration setting. Labels can be written in any supported language.

  Almost all labels can include embedded strings from the Strings repository of the Configuration dialog box. An embedded string takes the form of a decimal integer surrounded by square brackets, for example: [5]. Only attribute layout category labels cannot include embedded strings.

  See the section “The Strings Repository” in the *Infor Campaign Management Topic Implementation Guide* for additional information. for additional information.

- The **Developer Label** text box specifies the object name that is displayed to developers in Admin Manager. Developer Labels can be written in any supported language. The value that appears in this text box changes depending on the EpiMeta language configuration setting.

Using these parameters, an installation can have an EpiMeta database that uses English object names, an Admin Manager developer modifies the metadata in French, and end-users view the application in Japanese. See "Multi-Lingual EpiCenters" on page 78 for details about internationalization.

The Usage Tab

Dialog boxes that configure global objects, such as a measure or attribute, have a **Usage** tab that lists all of the objects that use that global object. Changes that you crate to a global object affect all of the

objects that appear in this tab. Infor suggests that you review the **Usage** tab before you confirm any changes that you might make to any global object.

Buttons

Admin Manager dialog boxes rely on a number of standard buttons to perform a variety of actions. Some of these buttons use icons to indicate the actions that they perform. Other buttons use labels.

Note: To see the name of a button that is labeled with an icon, click anywhere in the dialog box to ensure that the window is active, then let the mouse cursor hover over the button until a tool tip appears. You can also use Admin Manager Preferences dialog box to include a text caption on every dialog box tool button.

The following buttons and commands appear in a variety of dialog boxes within Admin Manager:

	Edit the selected item or rename it.
	Delete or remove the selected item. When the label for this button says, “Delete,” the item is deleted from the metadata entirely. When the label says, “Remove,” the item is removed from the parent object of which it is a component. However, the removed item remains available in metadata, typically through the Object Gallery.
	Create a new item or category.
	Duplicate the selected item.
	Undo the previous action.
	Move the selected item up or down within a list.
	Sort a list in either ascending or descending order.
	Indent the selected item left or right within a list.
	Refresh the display of the object gallery.
	Print the contents of a graphical display pane.
Apply	Apply the changes that you have specified in the dialog box without closing the dialog box.
Cancel	Undo changes in a dialog box since the last save.
Help	Open context-sensitive online help for the current dialog box.

Close	Close the dialog box. This button does not roll back changes to children of the object being edited. Usually, these children are shown in lists or trees under the parent object. Changes to those child objects cannot be rolled back in dialog boxes that have a Close button.
OK	Apply the changes that you have specified in the dialog box and close the dialog box.

The Object Gallery

Dialog boxes that use previously defined elements as building blocks display the elements that are available in the Object Gallery, which appears as a pane on the right-hand side of an appropriate tab within a dialog box. Admin Manager allows you to drag objects from the Object Gallery to the display pane of the tab in order to add those objects to an element that you are configuring.

The Object Gallery displays only objects that are appropriate for inclusion in a given dialog box or tab. For example, when you configure an extraction job, the Object Gallery displays only Extraction Groups and Extraction Steps.

Objects that you drag from the Object Gallery pane into the display pane are added to the element that you are currently configuring. Objects that you delete from the display pane are removed from the current element, although they remain available in the Object Gallery for later inclusion in the current element or in other elements.

You can add, remove, and edit an object if you select it in the Object Gallery pane. However, any changes that you create to an object in the Object Gallery are global in scope. Those changes apply to all other elements in which that object is used. Check the **Usage** tab of the dialog box before you delete or alter an object.

Note: Changes affects all occurrences of this object, not just the current dialog box.

Note: The Cancel button does not roll back changes that you create to objects in the Object Gallery.

"Figure 22: The Object Gallery" on page 75shows a measure layout that has been added to a web page from the Object Gallery.

Figure 4-10: The Object Gallery

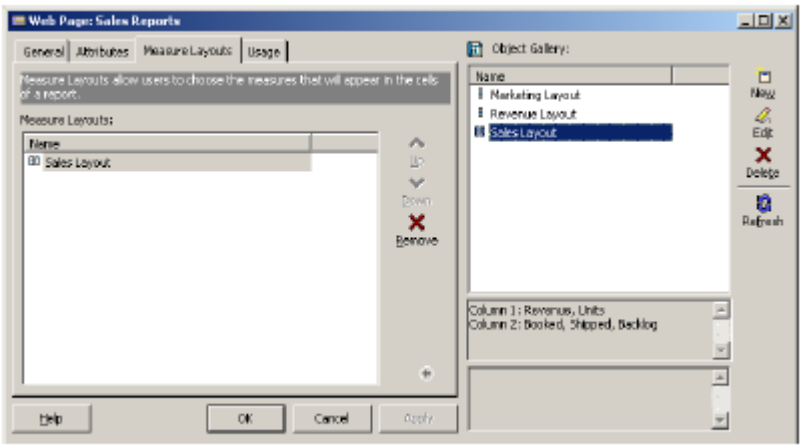


Figure 22: The Object Gallery

Data Flow Diagrams

You can use the graphical extraction-command builder to configure source-system data extraction. The graphical extraction-command builder displays the flow of data from the source system to EpiMart tables by means of a data flow diagram, as shown in "Figure 23: Data Flow Diagram" on page 76. See "Using the Graphical Extraction-Command Builder" on page 277, for more information.

Figure 4-11: Data Flow Diagram

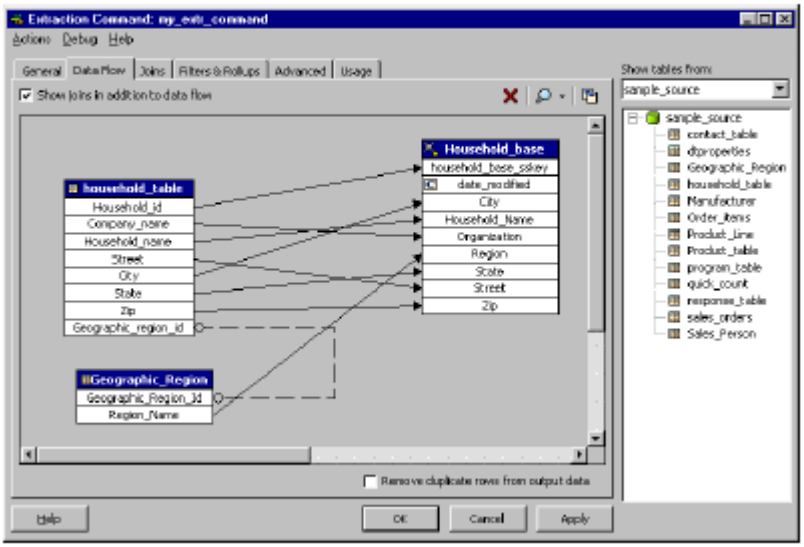
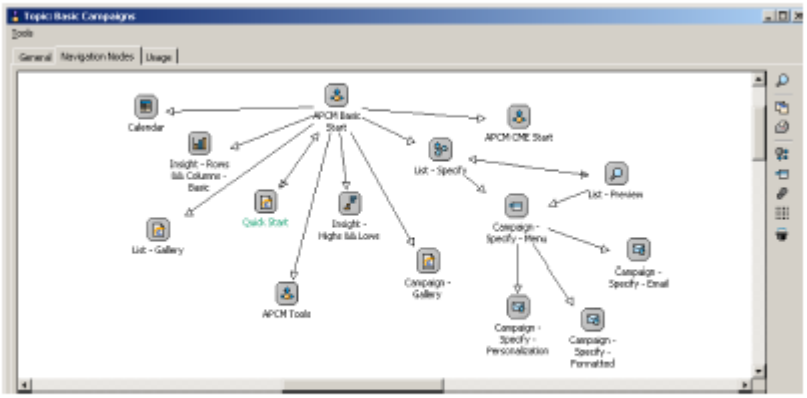


Figure 23: Data Flow Diagram

Link Diagrams

Link Diagrams are topics in which Infor Campaign Management applications are organized form networks of links that can sometimes appear complicated. To help keep track of the navigational paths within topics, the **Navigation Nodes** tab of the Topic dialog box displays a diagram of the nodes and links that have been configured within a topic. figure on page 15, shows an example of a link diagram, which includes links that point to and from a particular navigation node within a topic.



Propagating Elements Between EpiCenters

In Admin Manager you can use drag-and-drop to propagate elements between EpiCenters. Within the EpiCenter hierarchy in a single Admin Manager window, you can drag and drop items or folders (if the folder contains individual items, such as Measure Sets or Attributes) from one EpiCenter data mart to another.

Dragging objects between data marts is equivalent to exporting and importing those objects. See "Exporting and Importing Metadata" on page 429.

Online Help

Admin Manager includes a comprehensive, context-sensitive online help system. You can access online help:

- by clicking the **Help** button in any dialog box.
- by typing **F1** on your keyboard.
- in the **Help** menu, by selecting either **Contents** or **Index**.

Online Help Left-Hand Tabs

Admin Manager online help includes the following left-hand tabs to help you locate information:

- **Contents**

  The **Contents** tab displays books and pages that represent the categories of information in the online help system. When you click a closed book, it opens to display its content (sub-books and pages). When you click an open book, it closes. When you click pages, you select topics to view in the right-hand pane of the Help viewer.

- **Index**

  The **Index** tab displays a multi-level list of keywords and keyword phrases. These terms are associated with topics in the help system. Keywords are cross-referenced with synonyms to provide multiple ways for you to locate information.

  To open a topic in the right-hand pane associated with a keyword, select the keyword and then click **Display** . If the keyword is used with more than one topic, a Topics Found dialog opens so that you can select a specific topic to view.

- **Search**

  The **Search** tab enables you to locate all topics in the help system that contain words that you specify. You can choose to narrow a search by searching previous results only, finding similar words, or searching topic titles only. When the search is complete, a list of topics is displayed in order of relevance. Topics that include the most occurrences of the words that you specified are placed at the top of the list.

- **Favorites**

  The **Favorites** tab allows you to bookmark frequently-viewed Help topics. To add a topic to the **Favorites** tab, open the topic, select the **Favorites** tab, and click **Add** . To remove a topic from the **Favorites** tab, select the topic and click **Remove** .

Navigating Help Topics

You can navigate between topics in Admin Manager online help by following links embedded in the help text, or by clicking the **See Also** button located at the bottom of most topics. When you click **See Also**, a popup menu opens that displays a list of topics that are relevant to what is currently displayed in the right-hand pane. Select a topic from the popup to open it in the right-hand pane.

If you wish to return to a topic that you previously viewed during the session, click **Back** in the top-left corner of your help viewer.

For more information about Admin Manager online help, open the help viewer from Admin Manager and read the topics under **Using Admin Manager Help** in the **Contents** tab.

Multi-Lingual EpiCenters

Infor Campaign Management supports multi-lingual EpiCenters. You can create a multi-lingual EpiCenter by initializing an EpiCenter in one locale and then importing the labels for other locales from separate, translated .mdb files. Translated .mdb files are generated by the TUMs (Translate User Metadata) utility

that is provided with Infor Campaign Management. (Information on TUMs can be found in the TUMs Utility white paper on the Infor Support portal.)

When you implement a multi-lingual EpiCenter, there is always one master “from” language that is used to provide the baseline strings for translation to additional languages. Once the additional languages have been installed, however, all languages, including the original language, are equivalent. You can define objects in any language.

The Current Metadata Locale Setting

The language that is used to define EpiMeta objects is determined by the **Current Metadata Locale** setting. This setting can be changed in the **Languages and Locales** tab of the Configuration dialog box, or from the **EpiCenter** menu by selecting the **Choose: Current Metadata Locale** command. This selection applies only to the current EpiCenter and is displayed in brackets next to the EpiCenter’s name in the main navigation form.

Admin Manager automatically changes the metadata locale to the displayed value when the EpiCenter that is being edited is changed. If you edit more than one multi-lingual EpiCenter at a time, be careful not to confuse **Current Metadata Locales**.

The **Current Metadata Locale** setting is specific to an Admin Manager session. Different sessions of Admin Manager can edit the same EpiCenter while using different **Current Metadata Locales**.

A locale must be installed on disk for it to be available as a **Current Metadata Locale**. For example, you cannot change the current metadata locale to Spanish unless the supplemental files for Spanish have also been installed on the workstation, even if Spanish has already been imported into the EpiMeta.

Note: You can control the language that is displayed to end-users in the web-based Infor Campaign Management interface with the language_locale_key user preference, available in **Configuration > User Preferences**.

Installed, Logging, and Front-End Locales

The **Languages and Locales** tab of the Configuration dialog box displays the locales that have been imported into an EpiCenter. Locales can be marked as **Installed**, **Frontend**, or **Logging**.

- An **Installed** locale can associate a locale-specific label with any metadata object. Admin Manager also allows Installed locales to be selected as the **Current Metadata Locale** when editing EpiMeta.
- A **Frontend** locale can be selected by end-users in the web interface of Infor Campaign Management. Label entries for the locale must exist for all objects that the Infor Campaign Management Server uses (almost all metadata objects are used by the Infor Campaign Management Server). If labels for that locale do not exist, Scrutiny violations occur and the Infor Campaign Management Server does not start. Multiple locales can be marked as **Frontend** at any time.
- The **Logging** locale is used to create the Infor Campaign Management Server and EpiChannel logfiles. Labels for this locale must exist for all metadata objects. If labels for the locale do not exist,

Scrutiny violations occur and logging can produce meaningless messages. Only one locale can be set as the **Logging** locale at any time.

Full translations are not available for all locales. For example, there is currently no translation for French as a logging locale.

The initialization scripts and the **Load Builtin Labels** internationalization tool automatically set the **Installed** , **Frontend**, and **Logging** flags of the locales they load. The most recently loaded locale that is logging-capable is the default "Logging" locale, but this setting can be changed in Admin Manager. When an EpiCenter is initialized with a locale that does not support logging, English ('en') logging strings are automatically loaded. If you do not wish to use English logging, another locale that supports logging, such as 'ja', can be loaded and declared to be the logging locale.

Admin Manager, EpiMeta, and EpiMart Locales

The languages that are used in the EpiMeta database are independent of the languages that are used in the EpiMart. For example, EpiMeta can use French and Japanese to define objects while the EpiMart contains German and Spanish data. Though possible, this type of configuration is not recommended because it can confuse web users.

Likewise, the languages that are used in an EpiMeta or EpiMart are independent of the locales used in Admin Manager user interface. For example Admin Manager, display forms and menus in Japanese with an EpiMeta that does not have Japanese installed. Japanese forms can be used to enter metadata in other locales such as English or Korean. The locale used by Admin Manager's user interface can be specified in the Preferences dialog box with **Admin Manager's user interface locale**Admin Manager setting.

Note: Admin Manager translations are not available for all locales.

Creating a Multi-Lingual EpiCenter

To add additional languages to an initialized EpiCenter use the following procedure. Your database must be able to hold the code pages of each locale that is loaded. For example, to store Japanese characters it is common to use the UCS2 option on SQLServer and to use the UTF8 code page in Oracle and DB2.

Before starting work on your EpiCenter, ensure that your Windows host is correctly configured for the code page:

- In the Regional Settings control panel, the default (system) locale must match the code page that you are using for your EpiCenter.
- If you are using either DB2 or Oracle data stores, you must configure the locale environment variable on the server host. For DB2, the locale variable is DB2CODEPAGE, and the value must be the code page. For Oracle, the variable is NLS_LANG, and the value specifies both the locale and code page - for example, AMERICAN_AMERICA.WE8MSWIN1252.

Adding Additional Languages to an Initialized EpiCenter

1 Install the supplemental files for English (en) and for all other languages that you wish to use in your EpiMeta. For example, for a French EpiCenter that also includes German and Japanese labels, install the 'en,' 'fr,' 'de,' and 'ja' language packs. These files are available for download with the main Infor Campaign Management application.

Note: Installing supplemental files causes the files for a locale to be present on disk, but does not cause the locale to be present in any particular EpiMeta. Infor Campaign Management internationalization tools or import must be used to import the locale into an EpiMeta.

Note: The locales installed on disk can be found in the Add/Remove programs control panel tool, or by looking at the subdirectories of **<install root> > ConfigFiles > Metadata**.

2 Create a directory structure for your custom translation (TUM) files that is not located in your root Infor Campaign Management directory. By creating a directory outside of the root directory, you ensure that your custom translation files are not be overwritten when you perform Infor Campaign Management upgrades.

a Create a root translation directory, such as `D:/MyTranslations`.

b Create a sub-directory for each language pack that you wish to install. Name these directories with the standard Java two-letter language abbreviation code (for example, 'en' for English, 'ko' for Korean, 'fr' for French, and so on).

c Within each sub-directory, create another sub-directory named `TUM`.

The directory structure that you create must mirror the structure found in your original Infor Campaign Management root directory under **ConfigFiles > Metadata**.

3 Copy `Custom_AS_MTum.mdb` from **<root> > ConfigFiles > Metadata > en > TUM** to the new root directory in Step "2" on page 80. Place the file in **<newRoot> > <fromLocale> > TUM**, where fromLocale is the language from which all other languages are translated. This language is often English ('en'), and must match the language specified as the **Current Metadata Locale** for your EpiCenter. You can access this setting from the EpiCenter menu.

The TUM files are created automatically by the code in the MTUM file. You can rename the MTUM file as appropriate for your installation as long as you keep the `_MTUM` suffix. For example, `My Strings_MTUM.mdb`.

4 From the **Tools** menu, select **Internationalization > Load Built-in Labels** for all languages other than the one that is originally used for EpiCenter initialization. The TUMs process translates each string in the original ("from") language to each of these languages.

Note: Your master ("from") language must match the language specified as the Current Metadata Locale for your EpiCenter. You can access this setting from the EpiCenter menu.

5 From the **Tools** menu, select **Internationalization > Translate User Labels**:

a Supply the name of the directory that you specified in Step 2a on page 81. For example, `D:/MyTranslations`.

b Supply the name that you gave to your MTUM (TRANSSET) file in Step 3 on page 81. For example, if you renamed the file to be `MyStrings_MTUM.mdb` then enter **MyStrings** in this field. If you left the file as `Custom_AS_MTum.mdb`, enter **Custom_AS** in this field.

If the file structure that you created in Step 2 is complete, the translation process proceeds. .mdb files are generated in the directory for each locale.

6 Import the generated .mdb file for each non-master locale into your EpiCenter. Using the examples above, these files are located in `D:/MyTranslations/<locale>` and named `MyStrings_<locale>.mdb`. In the Import dialog box be sure to specify **Just Labels** in the Advanced Options area.

Note: If you do not specify Just Labels in the Advanced Options area of the Import dialog box, the foreign-language metadata replaces all objects in your existing EpiCenter. The only language that is stored in the EpiCenter is just imported, and associated objects can be deleted. For example, a replaced topic causes the deletion of all reports that are specified from the original topic

Translating Custom Strings

After initial translation, the number of translated strings depends on the amount of commonality between the built-in strings that are provided with a newly initialized EpiCenter and the strings that are contained in your EpiCenter. All strings that are not translated in the TUMs process are stored in the TUMs file that is located under your `ConfigFiles` directory.

You can view and edit the untranslated strings with the TUMs utility (see the TUMs Utility white paper on the Infor Support portal for details). Edits that you specify in the utility are permanently stored and used when you run Translate User Labels in the future. If you edit non-master-language strings directly in Infor Campaign Management, running Translate User Labels erases your work.

Translate User Labels updates both the TUM file with the latest strings in the master language of your EpiMeta, and updates the EpiCenter with the latest translations of non-master languages.

Note: Loading Korean or Japanese (double-byte character) labels using a machine with a Latin-based regional setting corrupts the double-byte characters and renders them unreadable.

Conversely, loading Latin-based characters on a Korean regional setting machine corrupts the Latin characters. In this case, all accents are stripped and labels are shown as readable words with only English characters.

Adding New Objects to a Multi-Lingual EpiCenter

If you add any new object to a multi-lingual EpiCenter (such as a user, measure, web page, and so on), the object is created with labels for the **Current Metadata Locale** language only.

There are two methods of defining labels for other languages:

- For every language that requires translation, change the **Current Metadata Locale** setting to the appropriate language and edit the object's labels in that locale. You can change the **Current Metadata Locale** setting from the **EpiCenter** menu of Infor Campaign Management.
- From the **Tools** menu, choose **Internationalization > Translate User Labels** to create import files from which the labels can be loaded. Follow the procedure in Steps 5 on page 81-6 on page 82 in "Adding Additional Languages to an Initialized EpiCenter" on page 81.

  Note: Because object edits leave metadata temporarily inconsistent, Infor recommends that you do not use Infor Campaign Management to create objects in a production EpiMeta other than by importing metadata for which all labels have been already created.

The production Infor Campaign Management Server does not require labels until it is refreshed or restarted, but unexpected power outages or machine reboots can disable your system.

Defining the EpiCenter Schema

5

This chapter describes how to use Admin Manager to define the data mart schema that you designed in "EpiCenter Schema Design" on page 27 .

Note: Do not copy and paste scripts, commands or code from this document. Line breaks and some other characters picked up when you copy from the PDF can cause errors in the pasted text.

Getting Started

Before you can set up your EpiCenter, you must complete the installation as described in the *Infor Campaign Management Installation Guide*. The installation procedure includes allocating space on the server for the EpiMeta, EpiMart, and EpiOp databases and installing the Infor Campaign Management software. The EpiMeta database defines the structure of your EpiMart data mart and the configuration of your Infor Campaign Management applications. You use Admin Manager to make changes to the EpiMeta database and to propagate those changes to the EpiMart database.

Setting Up an EpiCenter

To set up an EpiCenter, you need to do the following:

- First, register your database server with Admin Manager.
- Secondly, initialize an EpiCenter on that database server.

When you register an EpiCenter, Admin Manager displays that data mart until you explicitly unregister it. Every time that you start Admin Manager, all registered EpiCenters are shown with grayed-out icons. To connect to a data mart with a grayed-out icon (EpiCenter), double-click on the icon.

Registering a Database Server

1 Start Admin Manager.

During installation, the Infor program executable (epimgr.exe) is placed in the Infor directory (`C:\<Infor Campaign Management Install Directory>\bin`). Admin Manager can be selected from the **Start** menu:

Program Files > Infor > 10.1.0 > Admin Manager > Admin Manager

2 Choose **Register Server** from the **Server** menu to open the Server Properties dialog box.

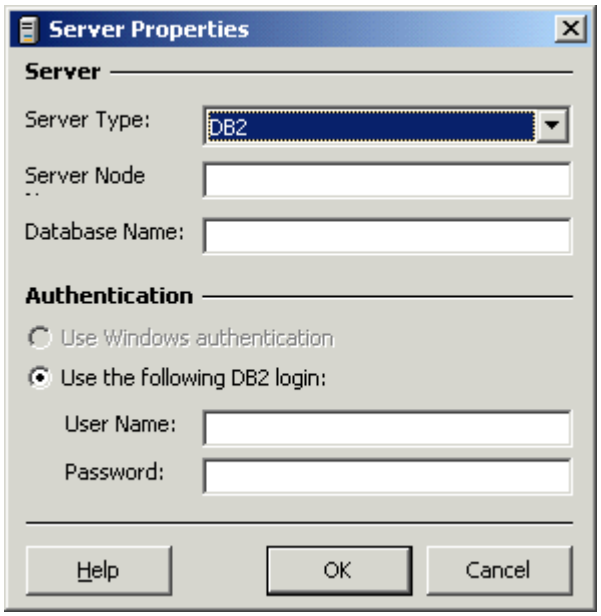


Figure 24: Server Properties Dialog Box

3 Select the **Server Type** from the list box.

If you've chosen SQL Server:

a If you've chosen SQL Server:

- Enter the **Server Name**.
- Choose **Windows Authentication** or **SQL Server** login.
- If you chosen SQL Server login, enter the administrator **User Name** and **Password** .

b If you've chosen Oracle:

- Enter the Oracle **Service Name** .
- Enter the Oracle administrator **User Name** and **Password** .

c If you've chosen DB2:

- Enter the **Server Node Name** .
- Enter the **Database Name** .
- Enter the DB2 administrator **User Name** and **Password**.

4 Click **OK** to register the server and open the **Choose EpiCenter** dialog box.

a If you have chosen either SQL Server or Oracle, you can select it from the list. If you have chosen DB2, enter the database name in the edit box.

Your database server's machine icon appears in the hierarchical tree structure. This server icon represents the server on which the EpiMeta database for this EpiCenter resides.

b If you have not previously created an EpiCenter on your database server, click **Cancel** to close the **Choose EpiCenter** dialog box.

c If the server already contains an initialized EpiCenter, you can use the **Choose EpiCenter** dialog box to select and open a pre-existing EpiCenter, as described in "Opening an Existing EpiCenter" on page 91.

Note: When you register a server for the first time, Admin Manager displays the Register EpiCenter dialog box after connecting to the server machine. You can either close this dialog box to continue with the initialization, or you can choose an uninitialized EpiCenter and initialize it later.

Unregistering an EpiCenter

You can unregister an EpiCenter by right-clicking the icon for the data mart and selecting **Unregister** from the pop-up menu. When you unregister an EpiCenter, the icon for the data mart is removed from Admin Manager's window.

Initializing an EpiCenter

1 Right-click the EpiCenter icon under the name of your server in Admin Manager's window and select **Initialize EpiCenter** from the pop-up menu. The Initialize EpiCenter dialog box opens.

Figure 5-2: Initializing Your EpiCenter

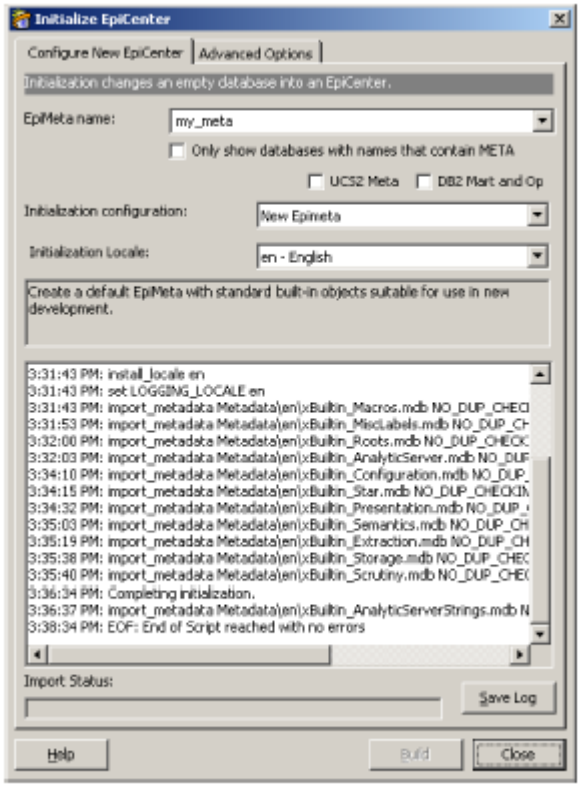


Figure 25: Initializing Your EpiCenter

2 In the **EpiMeta Name** list box, select the name of the new, empty database for the metadata that you created during installation (as described in the *Infor Campaign Management Installation Guide*).

3 If you are initializing an Oracle EpiMeta database, specify the EpiMeta password.

Note: You must use the same user name/password combination to initialize the EpiMeta, EpiMart, and EpiOp during creation of the database.

4 Select **UCS2 Meta** if you wish to initialize a database that can contain multi-byte (UCS-2) text. This option is only available with SQL Server EpiCenters.

5 In the **Initialization configuration** list, select the script that you wish to use to initialize your EpiCenter:

a Select **New** if you have never initialized an EpiMeta database in the selected Database.

b Select **Reinit** if you have initialized an EpiMeta database in the past and need to drop tables that currently exist.

c Select one of the **EM enabled** options if you are installing Email Marketing.

6 In the **Initialization Locale** list, select the language that must be used to initialize the names and labels of built-in objects in your EpiMeta.

Note that if you select a language other than the **Default Meta Locale** specified in the Preferences dialog box, the labels of your new EpiCenter's built-in objects are not visible in Admin Manager.

7 If desired, you can modify the physical properties for metadata tables and indexes in the **Advanced Options** tab of the Initialize EpiCenter dialog box (see "Figure 26: Initialize EpiCenter Dialog Box: Advanced Options Tab (SQL Server)" on page 89).

Figure 5-3: Initialize EpiCenter Dialog Box: Advanced Options Tab (SQL Server)

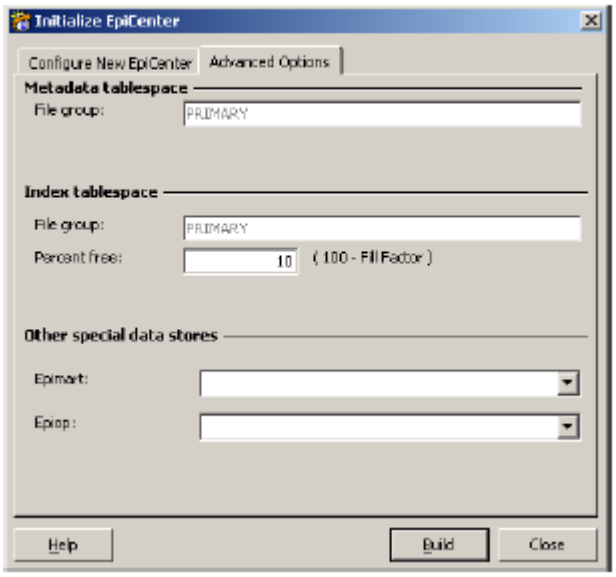


Figure 26: Initialize EpiCenter Dialog Box: Advanced Options Tab (SQL Server)

8 Click **Build** to start building a generic EpiMeta schema in the selected database. When you do this, the lower pane of the dialog box displays the status of the build, as shown in "Figure 25: Initializing Your EpiCenter" on page 88.

Initializing the EpiOp Database

After you initialize your EpiMeta database, you must initialize your EpiOp database. Before initializing your EpiOp database, verify that your EpiOp data store refers to the database that you are using as your EpiOp database. Do this by opening the Data Store dialog box for the EpiOp data store. See "Defining a Data Store" on page 205 for more information about the Data Store dialog box.

Figure 5-4: Initialize EpiOp Dialog Box

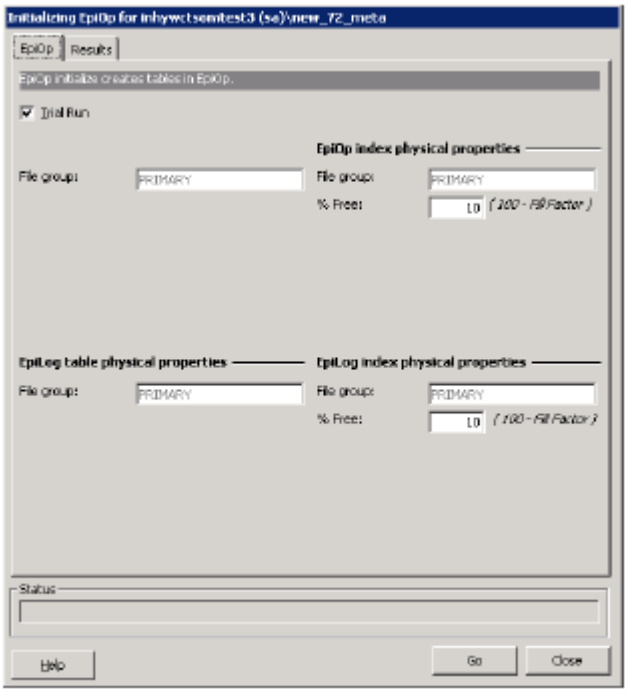


Figure 27: Initialize EpiOp Dialog Box

Initializing the EpiOp Database

1 Choose **Initialize EpiOp** from the **EpiCenter** menu for your EpiCenter. This displays the Initializing EpiOp dialog box.

In this dialog box, you can set physical object properties for EpiOp metadata tables and indexes (see also, "Physical Object Properties" on page 92, for more information about physical object properties). For most installations, the default settings are adequate.

Note: The settings that you configure here do not apply to the backfeed tables or saved lists in the EpiOp database. Physical object properties for backfeed tables are configured in the Physical Object Properties dialog box (see "Physical Object Properties" on page 92) and in the Object Properties tabs for the corresponding fact and dimension tables (see "Base Dimension Tables" on page 100 and "Configuring Object Properties" on page 137).

2 Select **Trial Run** and click **Go** to perform a trial run of the EpiOp initialization. No changes are made during the trial run. If the trial run is successful, deselect **Trial Run** and click **Go** to initialize your EpiOp database.

If you are importing metadata into your EpiCenter, EpiOp physical object property settings are also imported. This allows you to reuse an existing EpiOp database. If you wish to initialize a new EpiOp database for an existing system and reuse EpiOp physical object properties from an import file, be sure to do so after importing metadata into your EpiCenter.

Initializing the EpiOp generates a log file (init_epiop_summary.log). See appendix B, “Files Created By Infor Campaign Management,” in the Topic Implementation Guide for more information.

Opening an Existing EpiCenter

If an EpiCenter has already been initialized, you can open it in your Admin Manager window.

1. Register the database server on which your EpiCenter resides, if it is not already registered.
2. In Admin Manager’s window, right-click the EpiCenters folder icon and select Register EpiCenter from the pop-up menu.
3. In the Choose EpiCenter dialog box, select the EpiMeta for the EpiCenter.

   If you have chosen SQL Server or Oracle you can select the from the list. If you have chosen DB2 you enter the database name in the edit box. You can modify this data mart as appropriate for your site by following the instructions given in this guide for those items that you wish to change.

   Note: Admin Manager sometimes hangs when opening a DB2 metadata if the JDBC port number, as configured in the EpiMart or EpiOp datastore, is incorrect. If the incorrect port is an unused port, the driver can return “port out of range” and “unable to open socket” errors as well.

   To work around this issue, update the EpiMeta database directly using the following SQL statements (substitute the proper port numbers for the <JDBC port...> placeholders before running them). Be sure to commit the transaction after the SQL is run.

```
update db2_store

set server_port = <JDBC port for the EpiMart database>

where data_store_key = (select data_store_key from data_store where
built_in_type = 'EpimartStore')
```

```
update db2_store

set server_port = <JDBC port for the EpiOp database>

where data_store_key = (select data_store_key from data_store where
built_in_type = 'EpiopStore')
```

Physical Object Properties

You can configure default physical object properties for your EpiCenter in the Physical Object Properties dialog box. To display this dialog box, select Physical Object Properties from the **Tools** menu for your EpiCenter.

The Physical Object Properties dialog box allows you to configure vendor-specific database options for your EpiCenter. "Object Types and Views" on page 517 describes the types of objects for which you can configure these properties. For most installations, the default settings for these options are adequate. "Figure 28: Physical Object Properties Dialog Box (Oracle)" on page 92 shows the Physical Object Properties dialog box for an Oracle database server.

Figure 5-5: Physical Object Properties Dialog Box (Oracle)

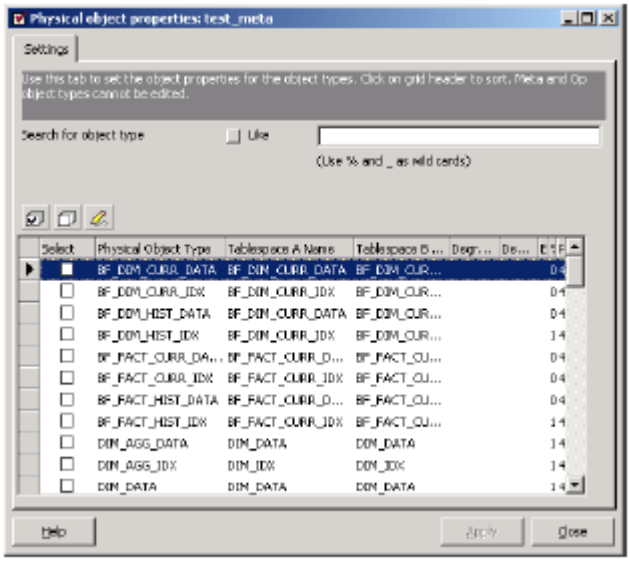


Figure 28: Physical Object Properties Dialog Box (Oracle)

For Oracle and DB2 database servers, it is possible to configure different tablespaces for objects in the A and B mirrors. See "Data Mart Mirroring: A and B Tables" on page 191, for a discussion of data mart mirroring.

Object Property Inheritance Hierarchy

You can specify physical object property defaults for an entire EpiCenter, an individual table, or a single partition within a table. When you create a new object, its object properties are inherited as follows:

- System defaults provide all settings. (Default settings are specified in the dialog box available from **EpiCenter > Physical Object Properties**.
- Dimension, fact, and external table settings override system default settings. (Table settings are available in the **Object Properties** tab of the Base Dimension or Fact dialog box).

- Oracle partition level overrides override all other settings. (Partition overrides are available by clicking **View Partition Details** in the **Object Properties** tab of the Base Dimension or Fact dialog box, or by editing the **Object Properties** tab of the Fact or Dimension Aggregate Instruction dialog box).

  The $$DEFAULT_PARALLEL_DEGREE macro specifies the **Degree Create** and **Degree Query** physical object properties at a level above the system defaults.Consequently, the value of this macro is only used if **Degree Create** or **Degree Query** are not specified in any other area of Admin Manager.

Oracle Object Properties

If your EpiMart database resides on an Oracle database server, the following properties can be configured for each type of object:

- Tablespace A Name . The name of the tablespace to which the objects in the A set of mirrored tables are assigned.
- Tablespace B Name . The name of the tablespace to which the objects in the B set of mirrored tables are assigned.
- Degree Create . The number of processes that are spawned when you are creating new objects of the specified object type.
- Degree Query . The number of processes that are spawned when you are querying objects of the specified object type.
- Extent Size in K . The number of kilobytes that must be allocated when an extent is created for objects of the specified object type.
- Pct Free . The percentage of block space that is allocated for expansion of existing objects.
- Pct Used . The percentage of block space that must be free before a block initially marked as “full” is reset to “available for inserts.”

SQL Server Object Properties

If your EpiMart database resides on an SQL Server database server, the following properties can be configured for each type of object:

- File Group A/B . The name of the SQL Server file group to which the objects are assigned for the A and B tablespaces (see "Data Mart Mirroring: A and B Tables" on page 191). For version 10.0 of Infor Campaign Management, this value is always `PRIMARY` .
- Pct Free (100 - Fill Factor) . The percentage of block space that is allocated for expansion of existing index objects.

DB2 Object Properties

If your EpiMart database resides on a DB2 database server, the following properties can be configured for each type of object:

- Tablespace A Name . The name of the tablespace to which the objects in the A set of mirrored tables are assigned.

  For MART_EPIAPP_DATA, turning off the dropped table recovery feature in the underlying tablespace for the A and B tables significantly enhances Application Server performance.

  Note: Prior to DB2 version 8.2, the dropped table recovery feature for tablespaces is turned off by default. With DB2 version 8.2 and later, the dropped table recovery feature for tablespaces is now turned on by default. Infor recommends that this feature be turned off.

- Tablespace B Name . The name of the tablespace to which the objects in the B set of mirrored tables are assigned.
- Pct Free . The percentage of block space that is allocated for expansion of existing objects.
- Partitioning and Replication . Specify whether to partition or replicate the table.

  Note: F or DB2, putting the data into multiple table spaces can improve performance, and allow more control when you are tuning database performance.

Partitioning

With DB2, you have the option of partitioning your fact and base dimension tables. Partitioning enables Infor Campaign Management to perform parallel updates and deletes, which can improve extraction and query performance. Each partition can sit on a separate node, and any data operation that can be performed using a single node (such as a query that joins a fact and a dimension that are partitioned on the same key) can be completed by multiple servers in parallel.

When you partition a base dimension, Infor Campaign Management uses the partitions in the underlying tablespace, using the dimension key to divide data between partitions. If you have not partitioned the underlying tablespace, it is treated as though a single partition exists. No error occurs, but the table is not partitioned as expected.

When you partition a fact table, the table partitioning is based on a dimension role that you select (rather than by date range, as with Oracle). This can help improve query and load performance in cases where the dimension and fact table are partitioned in such a way that operations are encapsulated within the smaller partitioned data sets. The sskey is used to divide the table data between partitions (The sskey is the source system key that allows one table to reference a record in another table).

Partitioning a DB2 Base Dimension or Fact Table

1. Ensure that the underlying tablespace is partitioned.
2. Select the **Partition** check box in the **Object Properties** tab for the table.
3. If you are partitioning a fact table, select the dimension role on which to partition the data. Normally, this can be a dimension that is also partitioned in a way that usefully intersects with the fact data.

Replication

Replicating a table copies the table data into multiple parts of a partitioned base tablespace in order to optimize joins in a multi-node environment.

Replicating a base dimension is most useful in cases where you have a small dimension that is used often but updated rarely, and a large fact table which is partitioned. When you join the fact to the replicated dimension (assuming that they are partitioned on the same key), each node has all the dimension data required for the join to its part of the fact table. This prevents data from being shuffled between nodes, creating a more efficient join.

Replicating a DB2 Base Dimension or Fact Table

1. Ensure that the underlying tablespace is partitioned.
2. Select the **Replicate** option in the **Object Properties** tab for the table.
   The fact or dimension table is replicated according to the partitions in the underlying tablespace.

DB2 Tablespace Recommendations

Note: DB2 table spaces which contain different mart and OP physical objects must have a large page size (32 KB). MART and OP databases must also have a system temporary table space with a large page size (32 KB).

Infor recommends separating out the following physical objects into separate tablespaces on DB2:

- Dimension Base Tables (DIM_DATA)
- Dimension Staging Tables (DIM_STAGE_DATA)
- Dimension Mapping Tables (DIM_MAP_DATA)
- Integer Mapping Tables (INTMAP_DATA)
- Fact Base Tables (FACT_DATA)
- Fact Staging Tables (FACT_STAGE_DATA)

Altering Object Properties

Modifications to the Physical Object Properties dialog box are applied only to new or rebuilt objects. On an Oracle database server, you can change some physical object properties of existing objects using the Alter Object Properties utility, as described in "The Alter Object Properties Utility" on page 391.

If you wish to propagate additional object properties to your EpiMart database for fact, dimension, or backfeed tables, you can do so by rebuilding the objects that you wish to change. You can rebuild objects in your EpiMart database by doing one of the following:

- Regenerate the schema (see "Generating the Schema" on page 161) with the **Force Rebuild of All Tables, lose all data in the mart** option selected. Do not select **Include Backfeed Tables** unless

you no longer need any of the data in the backfeed tables or you are able to back up and restore this data using your database administration tools.

Note: Regenerating the schema with the Force Rebuild of All Tables option destroys all data in your EpiMart database. You must re-extract all data after forcing tables to be rebuilt. If you include backfeed tables when forcing tables to be rebuilt, all backfeed data in your EpiOp database is also destroyed. After rebuilding any part of the schema with the Generate Schema utility, the next job that populates the rebuilt tables must be an initial load.

- Run an extraction job that rebuilds both mirrored copies of all tables for which you wish to change properties. You can do this by running an extraction job that includes the following sequence of steps. (Certain steps are repeated to force rebuild of both mirrored tables.)
  - Begin Extraction
  - Truncation steps for staging tables for all tables that are to be changed
  - Ignore New Fact Data semantics for all fact tables that are to be changed. These semantic steps must have the **Force Rebuild of Target Table** option selected in the Job dialog box.
  - Latest Dimension Value semantics for all dimension tables.
  - Builders. If the object property affects campaign or aggregate tables the appropriate builder must be run in rebuild mode.
  - A Commit extraction step
  - **Ignore New Fact Data** semantics for all fact tables that are to be changed. These semantic steps must have the **Force Rebuild of Target Table** option selected in the Job dialog box.
  - Latest Dimension Value semantics for all dimension tables.
  - Builders. If the object property affects campaign or aggregate tables the appropriate builder must be run in rebuild mode.
  - A Commit extraction step
  - End of Extraction

    Note: Altering physical object properties by running an extraction job is only adequate for tables and indexes that are rebuilt in extraction jobs. For example, staging tables are never dropped in extraction jobs, and streaming extractions never drop tables either. In both cases tables are truncated whether or not the force rebuild of target table option is selected.

See "Extraction Jobs," on page 85 for more information on configuring extraction jobs.

Setting Up an EpiCenter

A common strategy for setting up an EpiCenter is to proceed down the tree of folders in Admin Manager, starting with the Schema folder. You can, however, close and re-open folders and dialog boxes and continue to fine-tune your metadata before actually generating the schema, and you can regenerate your schema at any time.

After you have defined your site's metadata, you can use the **Generate Schema** command in the **EpiCenter** menu to implement these definitions in the EpiMart database. When you have generated

the schema in the EpiMart database, you can then extract data from your source systems into your EpiMart database.

The instructions in the remainder of this chapter describe how to set up a single EpiCenter. To create additional data marts, simply repeat the procedure that you used to create your first EpiCenter.

Defining Your Schema

The Schema folder consists of the Base Dimensions, Dimension Roles, Degenerate Dimensions, Fact Tables, and Relations sub-folders, as well as the Schema Observatory icon. You can use the dialog boxes in these sub-folders to define that metadata for the fact and dimension tables of your star schema.

Table and column names must use the ASCII character set and must be legal table and column names on your database platform.

Note: Do not use reserved words (such as SELECT or DESC) as column names in data mart tables. Using reserved words as column names can cause schema generation to fail. See your database server documentation for a list of reserved words for your database platform.

Schema definitions are recorded in your EpiMeta database. When you use the Campaign Management Adaptive Schema Generator to generate your schema (see "Generating the Schema" on page 161), the definitions in the EpiMeta database are used to define objects in the EpiMart database. If you change schema definitions, the adaptive schema generator can be used to propagate those changes to the EpiMart database.

The table below shows the effects of running Generate Schema after a particular operation is performed on your EpiCenter schema.

Table 15: Schema Operations

Operation	Action
Add a new base dimension	The table is created from scratch.
Delete a base dimension table	The table is dropped, and the dimension role columns of any fact tables that point to that base dimension table are deleted.
Rename a base dimension table	The table is renamed.
Add a new dimension column to an existing base dimension	The existing table is altered to include that column. All existing rows take the default value for that column.
Change the integer mapping status of a dimension column.	The dimension table is adapted without loss of data.
Delete a dimension column from a base dimension	The column is removed.
Rename a base dimension column	Treated the same as a deletion of the old column and an addition of a new one.

Operation	Action
Change partition instructions for a dimension table or change dimension table between partitioned and non-partitioned.	The dimension table is adapted without loss of data.
Change the memory lookup status of a base dimension.	Associated fact staging tables are adapted.
Add a new fact table	The table is created from scratch.
Delete a fact table	The fact table is dropped.
Rename a fact table	The fact table is renamed.
Add a new fact column	The table is modified to have the new column. All existing rows are set to a value of 0.
Delete a fact column	The column is removed.
Rename a fact column	Treated the same as a deletion of the old column and an addition of a new one.
Change partition instructions for a fact table or change fact table between partitioned and non-partitioned.	The fact table is adapted without loss of data.
Add a dimension role to a fact table	The fact table is adapted without loss of data.
Delete a dimension role	The fact table is adapted without loss of data.
Rename a dimension role	Treated the same as the deletion of the old column and an addition of the new one in all fact tables that reference the dimension role.
Add a new degenerate dimension to a fact table	A column for the degenerate dimension is added to the fact table. All existing rows are assigned the special value UNKNOWN. If you are using Oracle, adding degenerate dimensions to existing facts with a CHAR data type of less than seven characters in length produces errors. Use VARCHAR data types or CHAR (7 or more) to avoid the problem.
Delete a degenerate dimension	The column is removed from all fact tables that use the degenerate dimension.
Rename a degenerate dimension	Treated the same as a deletion of the old column and an addition of the new one.
Add a new relation	Nothing happens after running Generate Schema. You must run a MomBuilder job step in order to add a new internal table that links the dimension roles defined in the relation.

Operation	Action
Delete a relation	Nothing happens after running Generate Schema. You must run a MomBuilder job step for the internal relation table to be no longer available.
Add a new external table	The table is created from scratch.
Any change to an external table, including changes to columns	External tables are adapted when changed. New columns are added to, and removed columns are dropped from, the existing external table
Increase the data type length of a fact or dimension column (for example, from VARCHAR_25 to VARCHAR_50).	The column is altered without losing data.

See "Data Type Conversions" on page 504 for information on altering the data types of fact and dimension columns.

International Data

If you are using a localized version of Campaign Management software, you can store data in any supported character set in your EpiMart database. The names of tables and columns in your EpiMart database must use the ASCII character set, but the data in the tables can be in any supported character set.

When defining column data types, if your EpiMart database uses a multi-byte code page, keep in mind that `VARCHAR` data type lengths are measured in bytes. For example, a `VARCHAR_15` column can hold up to fifteen bytes of data. If the code page of your EpiMart database uses up to three bytes per character, then a `VARCHAR_15` column is limited to five characters, depending on the exact characters being stored. Be sure to choose a data type length that is sufficient to hold any data that you expect to extract from a source system.

Using Multiple Languages in your SQL Server Data Mart

If you wish to store data in multiple languages in a SQL Server data mart, use the NCHAR or `NVARCHAR` data types. `NCHAR` and `NVARCHAR` on SQL Server support Unicode (UCS-2) data.

Note: NCHAR and NVARCHAR are translated to the same data types as CHAR and VARCHAR for Oracle and DB2 data marts.

Base Dimension Tables

Base dimension tables are global objects that can be used throughout an EpiCenter. These are physical tables with dimension columns that contain attributes extracted from the source system. Dimension-column values are the attribute values that end users select as query criteria in a web page. See "Dimension Tables" on page 31 for more information.

Defining A Base Dimension

To define a new base dimension table, right-click the Base Dimensions folder icon and select **New Base Dimension** from the pop-up menu to display the Base Dimension dialog box.

Figure 5-6: Base Dimension Dialog Box: General Tab

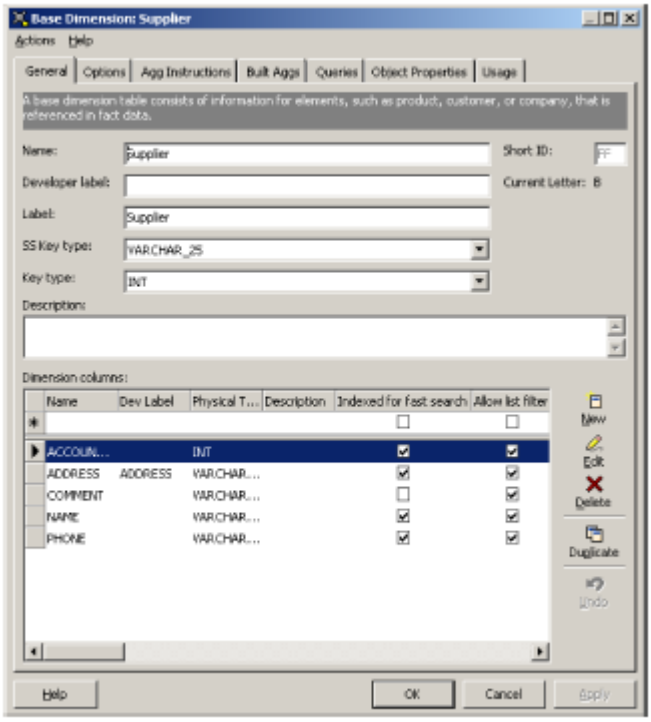


Figure 29: Base Dimension Dialog Box: General Tab

Defining a Physical Base Dimension Table

1 In the **Name** text box in the **General** tab, enter the name of the base dimension table as it should be defined in EpiMeta. The names of the physical database tables in the EpiMart database are formed by appending a suffix. All base tables have a suffix of the form `_0_N` (where N is A or B), and the system adds these suffixes to the names of the physical base dimension tables that it creates.

Note: The maximum length for the name of a dimension or fact table is 19 characters.

Note: If you rename a table, do not create another table with the same name until you have run Generate Schema. If you do not follow this procedure, you may get failures and even data loss.

Rename the schema table in one run of Generate Schema. Then recreate a new table with the original name in the next run of Generate Schema. That is:

- Rename the original object.
- Run Generate Schema.
- Create the new object.
- Run Generate Schema again.

2 In the **Developer label** text box, enter the name for the base dimension as it should be displayed in Admin Manager.

3 In the **SS Key Type** list box, select the source system key (sskey) data type. Normally, this is `VARCHAR_50` , but you may want to enlarge it. Certain database operations have a limit on their declared width, so avoid “over-declaring.”

4 In the **Key Type** list box, select the data type of the EpiKey used within Infor Campaign Management. This key is used to identify the base dimension to Infor Campaign Management components and should be able to contain a number at least as large as the number of rows in the dimension.

5 In the **Short ID** text box, enter a two-letter code that can be used for the names of tables that are derived from the dimension. This code must be unique within your EpiCenter. The default code is always unique.

6 In the **Options** tab, select the options that you wish to apply to the dimension. The table below describes each option.

Table 16: Options Tab - Dimension Table Dialog Box

Option	Description
This dimension is unmapped (fact tables refer to it with SSKeys)	Select this option to specify that fact tables should refer to rows in this table by the sskey value. In a mapped dimension, fact tables refer to a dimension row using an integer dimension key. A separate dimension mapping table associates this integer key with the corresponding sskey. The use of a mapped dimension can result in significant space savings in fact tables that join to the dimension, but it can impose the increased overhead of an additional join at query and extraction time. You can avoid this overhead by using an unmapped dimension. In general, you should only use an unmapped dimension if the dimension sskey uses a relatively small data type, since this sskey must be stored in every row of any fact table that joins to the dimension. Note the following:

Option	Description
	• Unmapped dimensions cannot use any versioning semantics. • Fact tables that join to unmapped dimensions may contain dangling references. With a mapped dimension, any fact staging rows that refer to a nonexistent dimension element are modified to refer to the UNKNOWN row in the base fact table. However, with an unmapped dimension, fact staging rows that refer to a nonexistent dimension element are not modified. Consequently, the corresponding base fact table rows contain dimension keys that do not join to rows in the base dimension table.
This dimension is non-versioned (it can never be slowly changing)	Select this option if you do not need to maintain references to older versions of data from this dimension with a semantic similar to Slowly Changing Dimensions. Selecting this option can reduce the width of the fact tables that are joined to this dimension and can result in faster campaign query performance.
Allow dimension fusion/fission	Select this option if this dimension will be used in fusion/fission semantics. A fused dimension can only use the Latest Dimension Value with Fusion/Fission or Latest Dimension Value, Preserve Fusion semantics. See "..." on page 263 for more information.
Allow EpiChannel to map SSKeys in memory	Select this option if you wish to bypass the step that maps dimension sskeys to dimension keys during fact semantics or dimension streaming semantics. You should select this option whenever you have a small dimension (with at most 500,000 distinct values in your columns). Otherwise, your extraction job may run longer, and EpiChannel may run out of memory.
Revision Percent	The percentage below which an incremental update can be attempted for this dimension table. If the number of new or changed rows in the difference tables are below this percentage of the current table's rows, then semantics attempt an incremental update. Leave this parameter blank to use the default percentage.

Option	Description
Lists can be made from this dimension	Select this option if you wish to make lists of the objects in this dimension, or if you wish to create a relation using this dimension.
Create seed tables for this dimension	Select this option if you wish to include seeds in this dimension. Seeds are dummy records that are included in campaigns that target members of the dimension. They can be used to verify that campaigns have been executed, or that fulfillment houses have not sold your campaign information to other vendors. See "Extracting Seed Data" on page 140 for details on how to set up seed tables in the EpiMart.
Include this dimension in backfeeds	Select this option if information from this dimension might be stored in backfeed tables. Backfeed tables store information about campaigns that have been run, including lists, list segments, treatments applied, and responses.
This dimension has inferred responses	Select this option if information from this dimension might be used in inference campaigns. Inference campaigns are designed to use available data to make inferences about the response to campaigns for which response data is not available.
Enable sampling for this dimension	Select this option if you wish to create statistical samples of the data in the main EpiMart tables. Sampled EpiMart tables are used whenever an end user requests fast approximate counts. When you select this option, you can specify the size of your sample in the Sample 1 row out of every text box. This number must be a power of ten. Be sure to use a sampling size that is statistically accurate. For example, if you have a organization dimension that contains only 10 organizations, sampling this dimension can lead to inaccurate approximate counts.
This dimension is the Real-Time dimension	Select this option if you want to enable the deployment of campaigns, lists, and data mart profiles based on this dimension to Real-Time. You can only mark one dimension as the Real-Time dimension.

Option	Description
This dimension is available for E-Mail Marketing	Select this option if you want to use lists based on this dimension for email campaigns. When this option is selected, you will be presented with the Choose Email Marketing Columns dialog box, which will allow you to add the EM-specific columns and attributes and designate the Email Marketing dimension role. EM-specific columns cannot be removed from a dimension that is available for Email Marketing. If an EM-enabled dimension is marked as no longer available for Email Marketing, the EM-specific columns and attributes are not automatically removed, but they can be manually deleted if desired.

Note: You cannot specify more Email Marketing dimensions than are configured in the Behavior/CME/MaximumEmailDimensions configuration setting (see "The Settings Tab" on page 147). This configuration setting should be set to the number of Email Marketing dimensions for which you are licensed.

Defining Dimension Columns

You define dimension columns in the **General** tab of the **Base Dimension** dialog box.

1. In the Dimension Columns pane, click **Add Column** .
2. In the **Dimension Column** dialog box, enter a name, developer label and a description for this column.

Figure 5-7: Dimension Column Dialog Box

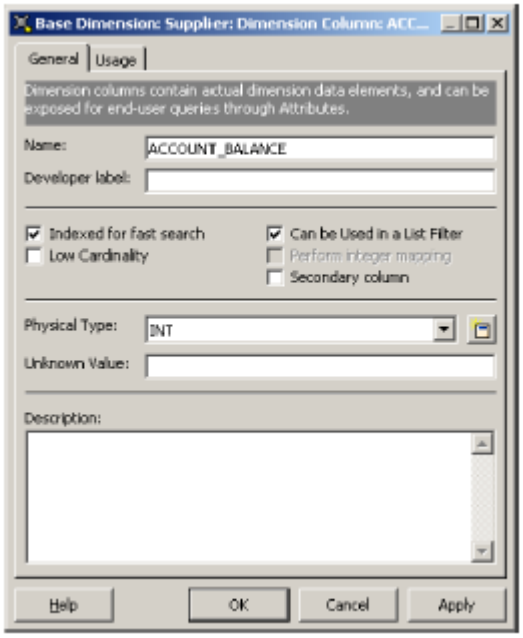


Figure 30: Dimension Column Dialog Box

3 Select the **Indexed for Fast Search** option to create an index on the column in the base table and in all dimension aggregates that have that column.

Create an index when you need to restrict Rows and Columns reports to a small subset of the overall data. A good guideline is if less than 20% of the base table is required.

4 Select **Low Cardinality** if the number of distinct values in the column is less than or equal to 10% of the total number of rows in the table. This flag is used to determine whether integer mapping should be allowed, and if bitmap indexes should be built on the base and thin dimension tables. See "Table 17: Interaction of Allow List Filter with Low Cardinality" on page 105 below.

5 Select **Can be used in a list filter** (**Allow List Filter)** to build an index on this column.

Table 17: Interaction of Allow List Filter with Low Cardinality

Allow List Filter	Low Cardinality	Action by MomentumBuilder
any	None	
X	**a** Create a composite index on the column and `dimension_key_REAL` column on the dimension base table. **a** Create a similar index on the dimension sampling table if sampling is enabled for this dimension.	

Allow List Filter	Low Cardinality	Action by MomentumBuilder
	The index is always a B-tree index regardless of database type.	
X	X	**a** Add the column to the dimension thin table. **a** Create a composite index on the column and `dimension_key_REAL` column on the dimension thin table. **a** Create a similar index on the dimension thin sampling table if sampling is enabled for the dimension. The index is always a bitmap index on an Oracle database and a B-tree index for other database types.

6 Select **Perform Integer Mapping** to associate a dimension-column value with a corresponding integer value. Integer Mapping can improve performance if you have a low-cardinality dimension column with values that require more than the amount of space required for an integer. This option can only be selected if **Low Cardinality** is selected.

DB2 restrict integer mapping to columns that are less than 1024 bytes in length. SQL Server restricts integer mapping to columns that are less than 900 bytes in length.

See "Integer Mapping" on page 34, for more information.

Note: A warning is displayed if you choose to perform integer mapping on a column with more values than the number specified in the **Behavior > EpiChannel > IntmapMaxValuesWarn key** in the **Settings** tab of the Configuration dialog box. The default value for this key is 500,000. An error is generated if the number of values surpasses the `IntmapMaxValues` key.

7 Select **Secondary column** if this is a secondary dimension column. Secondary dimension columns are typically for values that might use the Latest Dimension semantic. See "Primary and Secondary Dimension Columns" on page 34, for more information on primary and secondary dimension columns.

8 Select the data type from the **Physical Type** drop-down list. The actual data type that is used in the database tables depends on your database platform. The translation of physical type to database type is described in "Appendix C, “Data Type Values.”" on page 503

If you wish to create a new data type, click **New** . This action opens the **Data Types** tab of the Configuration dialog box. See "The Data Types Tab" on page 157 for information.

Note: If you define a SMALLDATE dimension column (to include a Customer_Birthday field, for example), your base dimension will snowflake (join) to the Date dimension (see "Snowflaked Base Dimensions" on page 36). If the date dimension is not populated with the dates that you wish to include in this field, the UNKNOWN date is substituted. Make sure that the Date dimension includes all dates that can be included in your data mart.

9 If desired, enter a default UNKNOWN value for the column. The initial UNKNOWN value is determined by the data type of the column (as specified in the **Data Types** tab of the Configuration dialog box).

To accept the initial value for the column, leave the **Unknown Value** field blank. Otherwise, you can enter a new unknown value.

a For character or string data types, enter the characters to include as an unquoted string that does not exceed the capacity of the column.

b For date types, enter a value of the form:

`mm/dd/yyyy`

Replace mm with the two-digit month. Replace dd with the two-digit day. Replace yyyy with the four-digit year.

c For numeric types, enter a value that does not exceed the capacity of the column.

For additional information, see also, "UNKNOWN Dimension Values" on page 34.

Defining Dimension Aggregates

Infor Campaign Management can pre-calculate aggregate fact values for selected base dimension column sets. These column sets, which are called dimension aggregates , represent a base dimension table in which one or more of the columns have been removed, and the rows have been collapsed onto one another to remove duplicates. When you define a dimension aggregate, you choose the columns in a base dimension table that are to be used for aggregation.

When defining dimension aggregates, be aware that there is a direct correlation between their size and the size of fact aggregates. A small change in the number of rows in a dimension aggregate can cause a large change in the number of rows in fact aggregates.

AggBuilder builds only those dimension aggregates that are included in fact aggregates.

The **Aggregate Instructions** tab of the Base Dimension dialog box shows all aggregates that have been defined for this base dimension. Click **Match Queries** to show the percentage of actual end user queries that have made use of this aggregate.

When defining the dimension columns for aggregates, keep the following in mind:

- Do not place two dimensions that have high cardinality (many distinct values) in the same aggregate if the data in those columns is independent. For example, City and Income may each have thousands of distinct values, and these values are not highly correlated with each other. Such a combination can result in an aggregate that is not much smaller than the original base dimension.
- Often one dimension column “drives” the aggregate. Aggregates are often based on a natural hierarchy in the data, and additional columns are higher-level roll-ups of the data in the driving column. For example, City drives County, State, Region, and Country.

Multiple aggregates are used when a base dimension table with many columns can be usefully aggregated in several different ways. Generally, reducing the number of columns in a dimension aggregate results in a smaller aggregate and speeds up response time for queries that use fact aggregates joined to that dimension aggregate.

For example, a customer dimension might have several fields related to customer geography (such as City, State, and Country) and several other fields related to demographics (such as age, marital status,

and years of education). You might then choose to define two dimension aggregates, one for the geographic attributes and one for the demographic attributes. You might also define smaller aggregates that use subsets of these column sets (for example, you might have an aggregate for only the Country value), or aggregates that overlap other aggregates that you have defined.

At query time, the Infor Campaign Management aggregate navigation mechanism chooses the smallest aggregate that satisfies the user's request. At query time, it is beneficial to have built many small, highly specialized aggregates. The penalty for building many dimension aggregates is increased extraction time and disk usage. The increase in extraction time can be substantial, so it is important not to define unneeded aggregates. Aggregates are best chosen based on actual query history. When your EpiCenter has accumulated a significant history of user queries, you should use query statistics (see "Dimension Query Statistics" on page 112) to analyze and fine-tune your aggregate definitions.

Defining a Dimension Aggregate for a Base Dimension Table

1 Click the **Aggregate Instructions** tab of the Base Dimension dialog box.

Figure 5-8: Base Dimension Dialog Box: Aggregate Instructions Tab

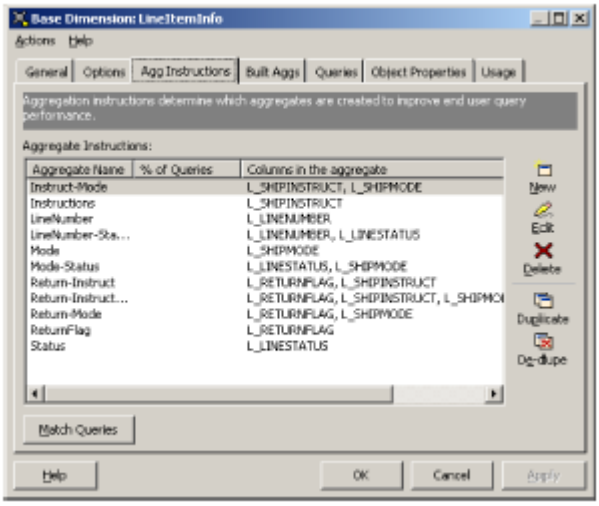


Figure 31: Base Dimension Dialog Box: Aggregate Instructions Tab

2 Click **New Agg** to display the Dimension Aggregate dialog box (see "Figure 32: Dimension Aggregate Dialog Box: General Tabl" on page 109).

Figure 5-9: Dimension Aggregate Dialog Box: General Tab

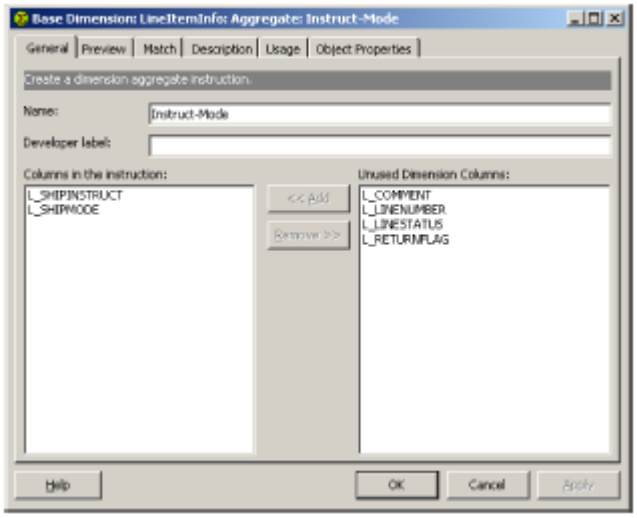


Figure 32: Dimension Aggregate Dialog Box: General Tabl

3 Enter the name and developer label for the aggregate in the **Name** and **Developer label** text boxes.

These names are logical identifiers only, and are not used during the building of the actual table in the EpiMart database. Instead, AggBuilder appends unique numbers to the base dimension table name for each new aggregate.

4 Add columns for this new aggregate by selecting them in the Unused Dimension Columns pane and clicking **Add**.

You can remove columns from the instruction by selecting them in the Unused Dimension Columns pane and clicking **Remove**.

5 Use the **Preview** tab (see "Figure 33: Dimension Aggregate Dialog Box: Preview Tab" on page 110) to preview the size of the aggregate that you are defining. The Preview SQL pane shows the SQL code that will be issued to determine the size of the aggregate. Click **Copy** to copy this code to the clipboard.

6 Click the **Preview** button in the **Preview** tab to display the size of the aggregate. The Preview Result pane shows the aggregate size in rows, the base dimension size in rows, and the aggregate size as a percentage of the base dimension size.

Figure 5-10: Dimension Aggregate Dialog Box: Preview Tab

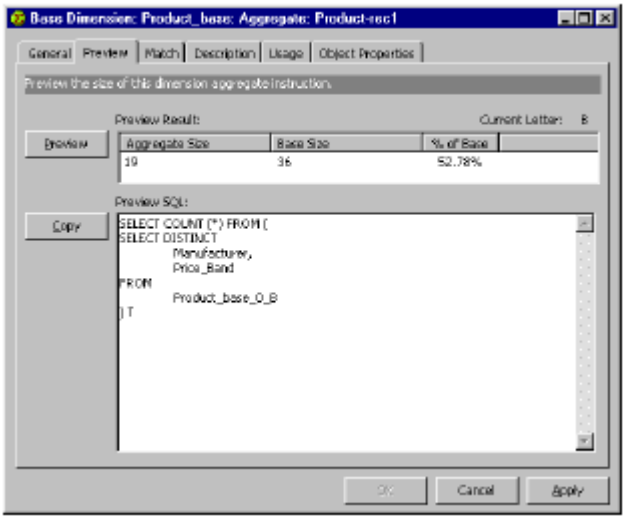


Figure 33: Dimension Aggregate Dialog Box: Preview Tab

7 Click the **Match** tab to compare the current aggregate definition to other existing aggregate definitions. The current aggregate instruction is compared to all aggregate instructions that match at least one column of the chosen column set. These dimension aggregates are shown in the Matching Aggregate Instructions pane grouped by the number of matching columns.

Expand a group and select an enclosed dimension aggregate to display the comparison in the Aggregate Comparison pane. This pane lists the columns in the current aggregate instruction and the selected comparison aggregate instruction, with matching columns displayed next to each other.

Figure 5-11: Dimension Aggregate Dialog Box: Match Tab

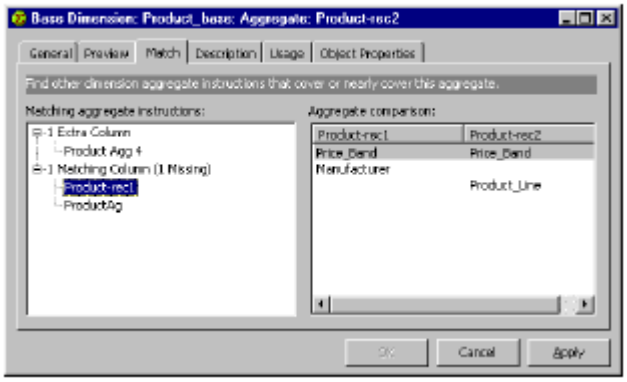


Figure 34: Dimension Aggregate Dialog Box: Match Tab

8 Enter a description for the aggregate instruction in the **Description** tab.

9 The **Usage** tab displays all fact aggregates that use this dimension aggregate.

10 You can configure database object properties in the **Object Properties** tab. See "Physical Object Properties" on page 92, for a description of the properties that can be configured here.

Duplicating an Aggregate

If an already created aggregate in the Aggregate Instructions pane of the Base Dimension dialog box is similar to one you plan to define, select it in the list and click **Duplicate** . Enter the new aggregate's name. Click **Edit** to modify it.

Note: The aggregate instructions that you enter in the Base Dimension dialog box are used the next time that AggregateBuilder is run. If a semantic instance has been applied to a dimension table, then new aggregates are built in the Next partition. If no semantic instance has been applied, then new aggregates are built in the Current partition. See "Swapping Mirrored Tables" on page 192, for a discussion of the Next and Current partitions.

Deleting an Aggregate

Click **Delete** to delete a selected aggregate instruction from your EpiCenter. This action automatically modifies all fact aggregate instructions that use this dimension aggregate instruction, possibly resulting in duplicate aggregates. You should dedup all fact aggregate instructions or run Scrutiny to check for duplicates after deleting a dimension aggregate.

Note that when you delete an aggregate instruction, an aggregate that is previously built from the instruction is not removed until additional extraction jobs have run (invalidating the aggregate), and you run the Purge utility.

Dimension Aggregate Browsing

You can use the **Built Aggregates** tab of the Base Dimension dialog box to view aggregates that have been built. Select either the A or B set of tables to browse. There are two copies of every dimension table, with suffixes `_A` and `_B`, containing "mirrored" copies of the table. For every dimension, either A or B is current at any time. One purpose of this table mirroring is to allow extractions to occur without disrupting the live usage of a system.

When you select the A or B tables, the built aggregates for those tables are displayed in the upper pane of the dialog box. If you select an aggregate in the upper pane, the dimension columns that are included in the aggregate are shown in the lower pane

Figure 5-12: Base Dimension Dialog Box: Built Aggregates Tab

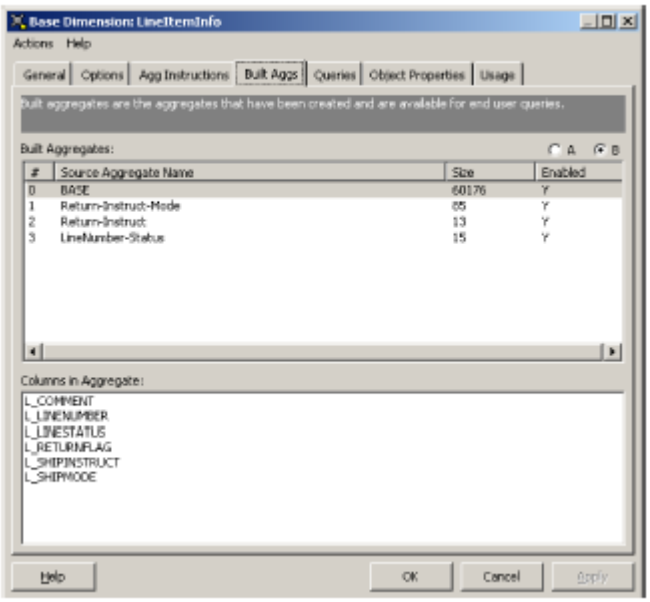


Figure 35: Base Dimension Dialog Box: Built Aggregates Tab

Dimension Query Statistics

Click the **Query** tab of the Base Dimension dialog box to display dimension query statistics (see "Figure 36: Base Dimension Dialog Box: Queries Tab" on page 113). Here you can view statistics about which combinations of base dimension columns have been queried and which of these queries could make use of dimension aggregates.

Note: Query statistics are only available if your EpiMeta and EpiOp databases reside on the same Oracle or SQL Server database server.

Figure 5-13: Base Dimension Dialog Box: Queries Tab

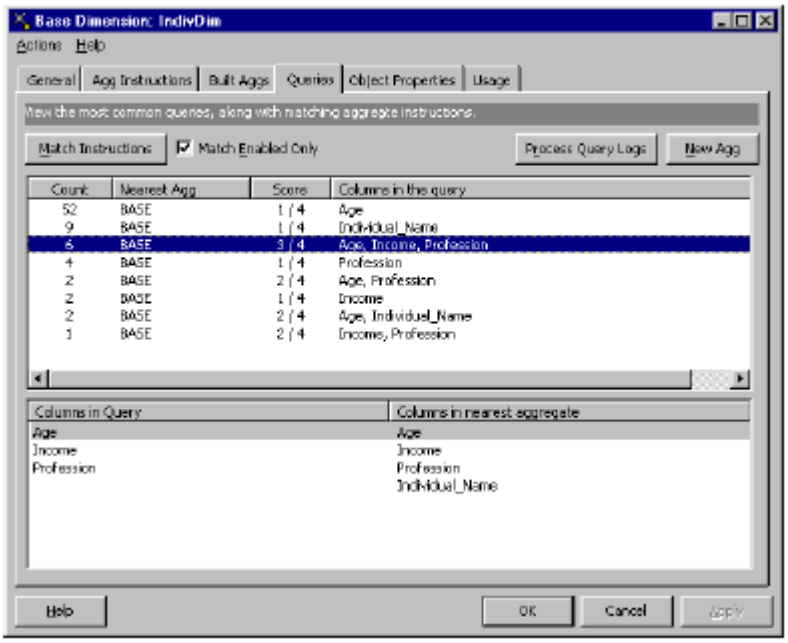


Figure 36: Base Dimension Dialog Box: Queries Tab

Query statistics are displayed in the upper pane of the dialog box. Each row shows statistics for a queried combination of base dimension columns. By default, only the number of queries and the queried columns are shown. Press the **Match Instructions** button to show statistics about matching aggregate instructions.

The following statistics are displayed in each row:

- Count: The number of queries that made use of exactly this set of columns in this base dimension.
- Nearest Agg: The dimension aggregate, if any, that best matches the column set.
- Score: If there is a matching aggregate, this column shows the proportion of dimension aggregate columns used for this query. For example, if three dimension columns were queried and the matching aggregate contains five columns (including the three queried columns), then **3/5** is displayed.
- Columns in the Query: The set of dimension columns that was queried.

Match Enabled Aggs Only is selected by default. Deselect it to include all aggregate definitions for consideration as possible matching aggregates. When it is selected, only dimension aggregates that are used by an enabled fact aggregate are matched to queries.

To view more details for a queried column set, select the corresponding row in the upper pane. When you select a row, the following details are displayed in the Query Details/Nearest Aggregate Details pane at the bottom of the dialog box:

- The Columns in Query column lists the queried dimension columns.
- The Columns in Nearest Aggregate column lists all of the columns of the nearest matching aggregate, if any. Matching aggregate columns are displayed adjacent to the matched query columns.

You can define a new dimension aggregate for any queried column set by selecting that column set and clicking **New Agg** . This displays the New Aggregate dialog box (see "Figure 32: Dimension Aggregate Dialog Box: General Tabl" on page 109), with the columns in the selected set already entered.

Click **Process Query Logs** to load or reload the query logs. When you click this button, the Process Query Logs dialog box (see "Figure 37: The Process Query Logs Dialog Box" on page 114) is displayed. In this dialog box, you can choose to reprocess all queries or to process only those queries that have not yet been processed. You can also set a query threshold. Queries that have been issued less often than the query threshold value are not processed.

Figure 5-14: The Process Query Logs Dialog Box

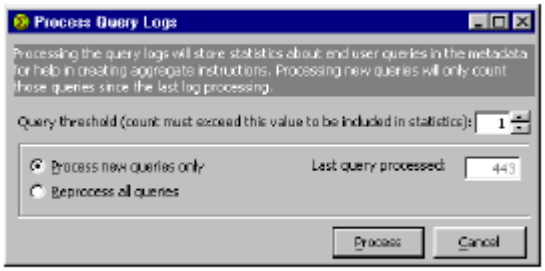


Figure 37: The Process Query Logs Dialog Box

Configuring Object Properties

You can configure vendor-specific physical object properties in the **Object Properties** tab of the Base Dimension dialog box. See "Physical Object Properties" on page 92, for a description of the properties that can be configured here and "Object Types and Views" on page 517 for a description of available object types. Any properties configured here override the object type defaults that are configured in the Physical Object Properties dialog box. See "The Alter Object Properties Utility" on page 391, for information on changing the properties of existing EpiMart objects.

On an Oracle database server with the Oracle Partitioning option, you can also define dimension partition properties in the **Object Properties** tab. Partitioning a dimension table can improve extraction performance by allowing parallel update and delete statements.

Note: In Oracle and DB2, when you partition a dimension table or index, only the dimension base and thin tables are partitioned. Dimension mapping tables are not partitioned.

Partitioning a Base Dimension

1. Select **Partition this Dim table** .
2. In the **Number of partitions** text box, select the number of partitions to use. For better data distribution across partitions, this number should be a power of 2 (for example, 2, 4, 8, 16, 32, and so on).
3. Click **View Partition Details** to edit partition-level object properties.

Note: Only partition-level overrides for tablespaces are used. Other partition object property settings are ignored.

Modifying the Date Dimension

By default, the Date dimension includes all of the columns shown in "Date Dimension Fields" on page 499 These columns cannot be changed or deleted. If necessary, you can add additional columns to the date dimension. You can populate these columns manually, or you can modify the date population routine to populate these fields automatically.

Note: clsDate.cls contains code for populating the vacation_day column, which indicates that a date is a holiday. This code uses US holidays, which means that you will need to modify this code if you need a different set of holidays. After modifying this code, build a new datepopulate.dll as described in this section.

If you add new date columns, then continue to follow the instructions in this section to populate them. Your code should be added to the code for the vacation_day column.

Also note that if you have custom date dimension columns, and have built a custom datepopulate.dll to populate those columns, then merge your code with the new code for the vacation_day column and rebuild datepopulate.dll.

To populate a new date column manually, simply use your database administration tools to add the appropriate values to the new column in the `Date_0` table of your EpiMart database.

To modify the date population routine, you must edit the DatePopulate.vbp Visual Basic project, which is located in the DatePopulate subdirectory of your Infor Campaign Management installation directory, and then use this project to rebuild the datepopulate.dll file that is used for date population.

DatePopulate.vbp contains a class called `clsDate`, which has three public subroutines that you can edit:

- `InitDatePopulate()`.
- This subroutine is called once before date population begins. You can use this subroutine to perform actions such as initializing variables, and opening log files.
- `FinishDatePopulate()`.
- This subroutine is called once after date population finishes. You can use this subroutine to perform such actions as closing log files and flagging errors.
- `DatePopulate(cCurrentRecord As Object, TDate As Date)`.
- This sub routine is called once for each row of date data that is written to the date dimension. The call is made before the row is written to the database but after the internal processing and calculation for that row is done. The data in the default date-dimension fields is available to this function through `cCurrentRecord`.
- `cCurrentRecord` contains the data for the current row of the date dimension that is being populated. Any data changed or sent back using `cCurrentRecord` will be written to the database. If the data in a field is not changed, then the field is updated with the default value set for that field. `TDate` is the current date value.

- While you can change values in the built-in date-dimension columns, Infor recommends that you avoid doing so. You should use this function to set values for all additional date-dimension columns that you have defined. The column values can be in any order.
- The `GetField` method returns the value of a field as a Variant that can be assigned to a string variable:

```
CCurrentRecord.GetField(fieldName As String)
```

- The `SetField` method changes the value of a field:

```
CCurrentRecord.SetField(fieldName As String,   Optional fieldVal As
Variant,   Optional isStringLiteral As Boolean)
```

Note: .Net is not supported in Infor Campaign Management 10.1.0. Use VB6 to update and generate DatePopulate.dll.

When you have made all desired changes, use Visual Basic (version 6) to **Make** a new version of the datepopulate.dll library. Before you build a new version of the library, you must ensure that the project is configured to produce a compatible binary. In Visual Basic, do the following:

- Select **DatePopulate Properties** from the **Project** menu to display the Properties dialog box.
- Click the **Component** tab.
- Under **Version Compatibility** , select **Binary Compatibility** and enter the path of the datepopulate.dll file that was installed in the bin directory by the Infor Campaign Management installer.
- Click **OK** to finish configuring project properties.

  After the properties have been properly configured, build a new version of the datepopulate.dll library. Replace the copy of datepopulate.dll in the bin subdirectory of the installation directory for your Infor Campaign Management instance with the new version and use the Windows regsvr32.exe utility to register the new library.

  If you change the structure of your date dimension, you must repopulate the date dimension in order to propagate your changes to the EpiMart database (see "Generating the Schema" on page 161).

Note: Repopulating a dimension causes related aggregates, indexes and MomentumBuilder tables to be rebuilt. You must run a builder job to repair the dropped aggregates and Momentum accelerator tables.

Dimension Columns for Data Mining Web Pages

Influences, Community Clusters and Bayesian Classifier web pages depend upon a primary dimension . The primary dimension is the business entity about which one wants to make a prediction or in which one wants to find clusters.

Influences and Bayesian Classifier allow end users to determine the effect of dimension attributes on facts, or on other attributes. Community clustering allows users to find relationships among the attributes of a dimension. You can also use facts for clustering and as influencers by generating a list based on these facts and then using that list as a list membership attribute. The use of lists, however, can slow down query response time.

To give end users quick responses to such data mining queries based on fact data, you can define additional dimension columns for that data. For example, if users might issue data mining queries based on whether a customer bought Product X, you could add an additional column `bought_X` to the Customer base dimension table. You can then extract the appropriate fact data into that column for every member of the dimension table.

If you choose to include fact data in dimension columns in this way, you should keep the following considerations in mind:

- Rapid changes in these special dimension columns can affect the accuracy of Community Clusters and Influences results. In a slowly changing dimension, a change in the value for any column results in the addition of a new row to the dimension table.
- For example, if a customer who has not previously done so buys Product X, a new row is added to the dimension table to reflect that purchase. The dimension table now has two rows for the customer: one in which Product X was purchased and one in which it was not.
- If another customer who has purchased Product X changes occupation, then a new row is added to the dimension table to record that change. The dimension table now has two rows for this customer, both of which indicate a purchase of Product X. Since Influences and Community Clusters look at all rows of a dimension table, these kinds of changes can lead to anomalous results in some circumstances.
- Removal of transaction data from the source database can affect the values in these special dimension columns. For example, assume that purchase data is removed from the source database after one year. If you are using a semantic that updates dimension data (such as latest dimension value or slowly changing dimensions), then the data for a customer may be incorrectly updated after the old sales data has been removed from the source system. If a customer purchased Product X eleven months ago, then a new entry for that customer will indicate that Product X was purchased. A month later, the source system no longer shows the purchase, so the information for this customer may be incorrectly revised to indicate that the customer did not buy Product X.

Dimension Roles

A row in a fact table can have several foreign keys that point to the same base dimension table. For example, there can be keys for Bill-to and Ship-to addresses that refer to the same Customer table. In such a case, the dimension is used in two different roles. A dimension role maps a fact foreign key to the associated physical base dimension table.

Whereas a base dimension is a physical database table, a dimension role is a logical entity that defines the meaning of the dimensions to the EpiCenter. See "Dimension Roles" on page 38 for more information.

Note: Admin Manager automatically creates a dimension role of the same name for each base dimension.

Defining a Dimension Role

1 Right-click the Dimension Roles folder and select **New Dimension Role** to display the Dimension Role dialog box (see "Figure 38: Dimension Role Dialog Box" on page 118

Figure 5-15: Dimension Role Dialog Box

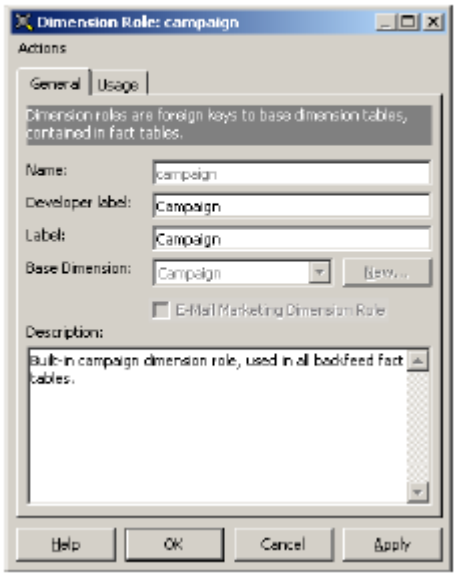


Figure 38: Dimension Role Dialog Box

2 Enter a name, developer label, and label for the dimension role. `Name` is the name of the dimension role as defined in EpiMeta. `Developer Label` is the name of the dimension role as displayed in Admin Manager, and `Label` is the name of the dimension role as displayed to end users.

3 Select the name of the base dimension table to be associated with the dimension role from the **Dimension Name** drop-down list box. Click **New** if you wish to create a new base dimension.

4 In the **Description** text box, enter an optional description of the dimension role for your reference.

5 To designate this Dimension Role as the Email Marketing dimension role for an Email Marketing dimension, check the **Email Marketing Dimension Role** box. The Email Marketing dimension role is the dimension role that uses in Email Marketing facts. To facilitate Infor Email Marketing campaign analysis, this should generally be a dimension role that joins to facts that are used in defining Infor Email Marketing campaigns. Note that this box is only enabled for Infor Email Marketing-enabled dimensions.

6 From the **Action** menu, click **Create Attributes** to display the Attribute Creator wizard for this dimension role. The Attribute Creator wizard is described in chapter 5, “Configuring Data Elements,” in the *Topic Implementation Guide*.

7 Click **OK** . The dimension role icon is added to Admin Manager’s tree.

Degenerate Dimensions

A degenerate dimension can be thought of as a dimension with a single column. Data for a degenerate dimension is stored directly in a text field of a fact table, rather than in a distinct table that is referenced by a foreign key in the fact table. For example, order, invoice, and bill of lading numbers have no corresponding dimension attributes. The only data of importance is the number itself; all of the other information of value is extracted into other dimension tables. See "Degenerate Dimensions" on page 39 for additional information.

Defining a Degenerate Dimension

1 Choose **New Degenerate Dimension** from the **Objects > Schema** menu to display the Degenerate Dimension dialog box.

Figure 5-16: Degenerate Dimension Dialog Box

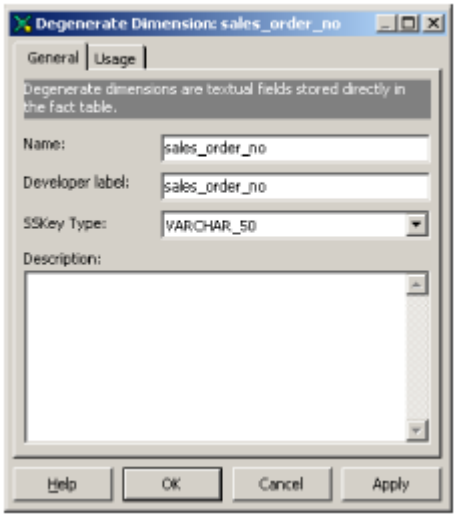


Figure 39: Degenerate Dimension Dialog Box

2 In the **Name** text box enter the name of the degenerate dimension as it must be defined in EpiMeta.

3 In the **Developer label** text box, enter the name of the degenerate dimension as it should be displayed in Admin Manager.

4 In the **SSKey type** list box, select the source system key (sskey) data type. Valid SSKey types are:

- CHAR and NCHAR
- VARCHAR and NVARCHAR (up to 2000 characters)
- INT, BIGINT, SMALLINT, and TINYINT
- NUMBER
- DECIMAL

5 Enter a description, if desired, and click **OK**.

Fact Tables

A fact table is a physical database table that holds numeric data. Fact-table rows also represent the intersection of a series of dimension keys. A fact table consists of dimension role foreign keys, degenerate dimension keys, and fact columns . The fact columns contain the numeric data in the fact table. See "Fact Tables" on page 38 for additional information.

Defining a Fact Table

1 Right-click the Facts folder icon, and select **New Fact Table** to display the Fact Table dialog box (see "Figure 40: Fact Table Dialog Box: General Tab" on page 120).

2 In the upper pane of the **General** tab, **Name** text box, enter the fact table's name. The maximum length is 19 characters.

Figure 5-17: Fact Table Dialog Box: General Tab

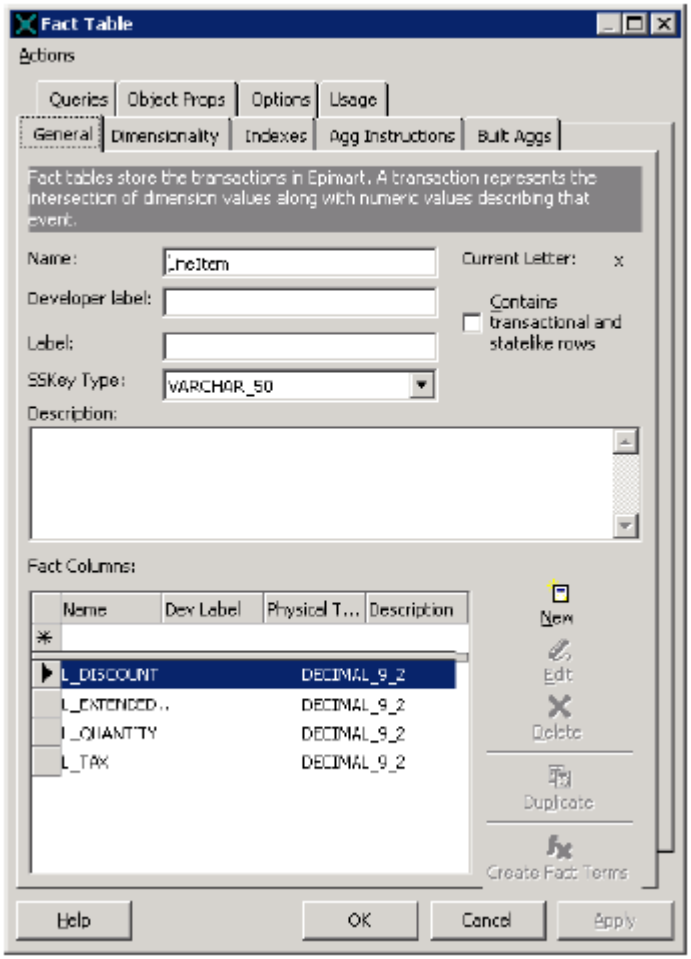


Figure 40: Fact Table Dialog Box: General Tab

Note: If you rename a table, do not create another table with the same name until you have run Generate Schema. If you do not follow this procedure, you can get failures and even data loss.

Rename the schema table in one run of Generate Schema, then recreate a new table with the original name in the next run of Generate Schema. That is:

- Rename original object.
- Run Generate Schema.
- Create the new object.
- Run Generate Schema again.

3 In the **Developer label** text box, enter the name of the fact table as it must be displayed in Admin Manager.

4 Select the source system key (sskey) data type from the drop-down list box.

5 Select the **Contains transactional and statelike rows** option if the fact table contains both transactional (facts indicating some discrete event, such as a shipment) and statelike (facts indicating the current state of some business quantity, such as inventory level) rows.

When you select this option, the Infor Campaign Management application inserts the `process_key` column into your fact staging table. This column indicates whether a row contains a transactional or statelike fact.

Note: If a table has both transactional and statelike data you must not use the same sskey to refer to both types of facts.

6 Enter a textual description of this fact table for your reference.

7 In the **Options** tab, specify additional options for the fact table. These options are summarized in this table:

Table 18: Options Tab - Fact Table Dialog Box

Option	Description
Revision Percent	The percentage below which an incremental update can be attempted for this fact table. If the number of rows in the difference tables are below this percentage of the current tables' rows, then semantics attempt an incremental update. Leave this parameter blank to use the default percentage.
Roll off number of units	If you do not wish to keep historical data that is older than a specified value, you can configure a fact table to roll off outdated data, as described in "Fact Roll-Off" on page 176. Use this option to specify the number of days, weeks, months, or other unit of time (specified in Roll off granularity) that data must be saved before being deleted. Select 0 if you do not wish to roll off fact data. Rolling off data from your fact tables allows you to delete old data from your fact table that you no longer need.

Option	Description
Roll off granularity	The unit of time with which you can specify how long you want to preserve fact data before it is deleted.

Fact Columns

Fact columns contain the actual customer numeric data, such as net_price or number_units . You can use the Fact Columns pane of the **General** tab of the Fact dialog box to define a fact column.

Defining Fact Columns

1 Click **New** to display the Fact Column dialog box ("Figure 41: Fact Column Dialog Box" on page 122).

Figure 5-18: Fact Column Dialog Box

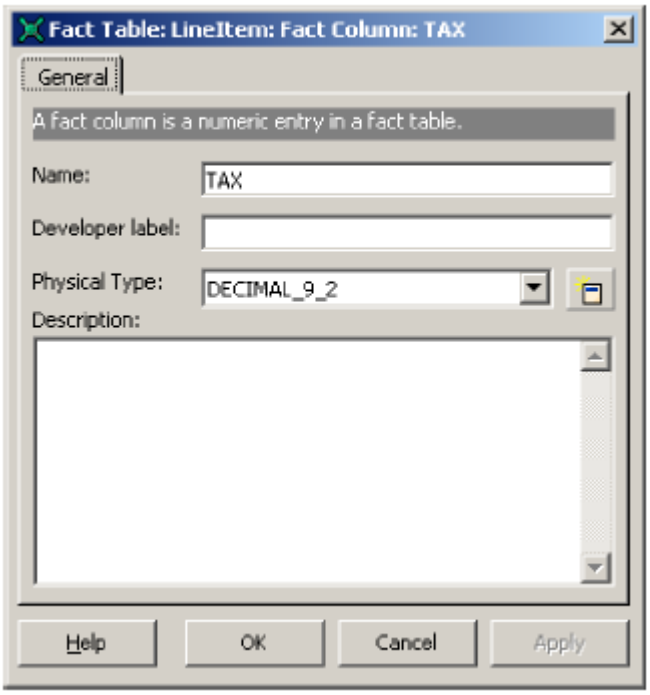


Figure 41: Fact Column Dialog Box

2 In the **Name** text box, enter the name of the fact column as it must be defined in EpiMeta.

3 In the **Developer label** text box, enter the name of the fact column as it must be displayed in Admin Manager.

4 In the **Physical Type** list box, select the data type of the column. The data type that you select is replaced by its associated database type when the database tables are built. The translation of data type to database type depends on your RDBMS platform. See "Appendix C, “Data Type Values”" on page 503 for descriptions of physical types.

   If you wish to create a new data type, click **New**. This action opens the **Data Types** tab of the Configuration dialog box. See "The Data Types Tab" on page 157 for information.

   Note: If a fact table contains a column of type TINYINT, then any semantic with a Row Type of Statelike, Transactional/Statelike, or Pipelined cannot be used with that fact table. In addition, any semantic with the Force Close option set to Yes cannot be used with that fact table.

5 Enter a description, and click **OK** to return to the Fact dialog box. The new fact column appears in the list box.

   a To remove a column from the fact table, select the column in the Fact Table dialog box and click **Delete** . To modify a column description, select the column and click **Edit** .

   b To automatically create basic fact terms for columns in your fact table, select the columns and click **Create Fact Terms** . (See “Fact Terms” in chapter 5 of the Topic Implementation Guide for information.)

Dimensionality in Fact Tables

In a star schema, the fact table is in the center of the star, and the points that radiate outward are the dimensions and degenerate dimensions. Dimension tables are referenced by foreign keys in fact tables. The values for degenerate dimensions are stored directly in the fact table.

You use the **Dimensionality** tab of the Fact Table dialog box (see "Figure 42: Fact Table Dialog Box: Dimensionality Tab" on page 124) to assign these keys to a fact table.

Figure 5-19: Fact Table Dialog Box: Dimensionality Tab

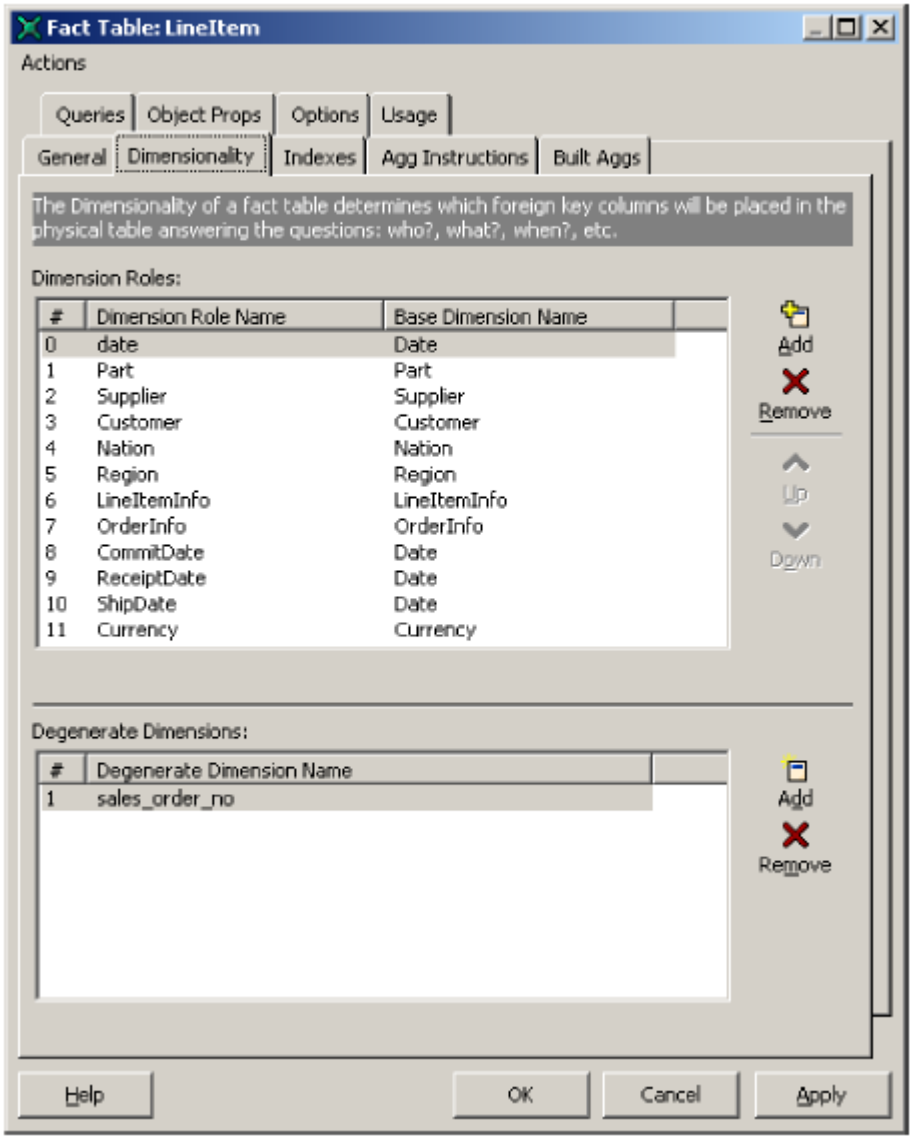


Figure 42: Fact Table Dialog Box: Dimensionality Tab

Assigning a Dimension Role Foreign Key

1. In the Dimension Roles pane, click **Add** to open the Choose dialog box.
2. Select one or more dimension roles. Or, click **New** to open the Dimension Role dialog box, where you can create a new dimension role.
3. Click **OK** to add the selected dimension roles.

   To remove a dimension role that you have added, select the dimension role and click **Remove**.

Note: Previous releases of Infor Campaign Management required the Generate Schema utility to rebuild any fact table after dimension roles are added or deleted, losing all data in the table. The Generate Schema utility can now adapt fact tables without loss of data after dimension roles are added or deleted.

Transaction Type and Date Dimensions

When defining fact tables, remember that the Transtype (transaction type) and Date dimensions occur in every fact table.

The transtype identifies a “slice” of the fact table that contains a single type of transaction. Rather than using multiple transtypes, you can configure a separate fact table for each transaction type. The advantages of having multiple transtypes are that measures that require two transaction types, (such as `BOOK` and `BOOK_RETURN`) need to issue only one query, and that interrelationships among transtypes can be managed by Infor Campaign Management semantic templates. The advantages of multiple fact tables are that queries that need only one transtype have fewer rows to navigate, and that aggregate tables are smaller.

All fact tables have `date_key` as a foreign key to the Date dimension. In addition to the built-in date dimension role that is used for `date_key`, you can define additional dimension roles that refer to the Date base dimension. If you define such additional dimension roles, keep in mind that CRM Infor Campaign Management always uses the built-in date dimension role for internal calculations (such as backlog calculations). In a fact staging table, the column name for the foreign key to a dimension role that references the Date dimension is `dimrolename_key`, where `dimrolename` is the name of the dimension role. Notice that the suffix for this foreign key is `_key`, rather than `_sskey`.

Note: Because of the special usage of `date_key`, it cannot be used as an attribute directly. Use day_name as an alternative.

As with all dates in the system, only the day-level granularity can be used for these dimension roles. You can use the built-in `$$TO_EPIDATE` macro to convert datetime values to the proper format.

Fact semantics treat additional dimension roles that refer to the Date dimension like dimension roles that refer to user-defined base dimensions. If the value for such a dimension role changes in some fact, then that fact are de-booked and re-booked with the new values by any semantic that allows the changing of facts. All columns in the Date dimension are available for querying. The Infor Campaign Management Server sorts these columns in the same manner as the built-in date dimension role.

Including Degenerate Dimensions

To include degenerate dimensions in the dimensionality of a fact table, take the following steps:

1. In the **Dimensionality** tab, Degenerate Dimensions pane, click **Add** and select one or more degenerate dimensions from the Choose dialog box.

   Clicking the **New** button in the Choose dialog box opens the Degenerate Dimension dialog box for creating a new degenerate dimension.

2. Click **OK** to add the selected degenerate dimensions.

Fact Indexes

In general, aggregates satisfy high-level queries while indexes satisfy highly selective queries. Query drill-downs on a web page transition from broad to selective. In terms of aggregates and indexing, the sequence of a drill-down transition from top to bottom are as follows:

- A small aggregate (for example, Business Unit equals Copiers).
- A larger aggregate (month equals January 1999).
- An index on a base dimension table (drill down by Customer Name equals John Doe).

  Aggregate tables can also be accessed by means of indexes. Aggregate tables are indexed in the same way as their base tables on those columns that still exist in the aggregate. For example, if a Sales fact table is indexed on (`product` , `household`, `customer`) and an aggregate is built with only the `product` , `household`, and `date` dimension roles, then that aggregate is indexed on (`product` , `household`). AggBuilder does not build duplicate indexes if two fact-table indexes result in the same aggregate index. For example, if the Sales fact table is also indexed on (`product` , `sales_person`, `household`), AggBuilder does not build a duplicate index on (`product` , `household`) in the aggregate table.

Defining a New Index

You can use the**Indexes** tab of the Fact Table dialog box to create a new index.

1 In the Indexed Columns pane, click **New** .

Figure 5-20: Fact Table Dialog Box: Indexes Tab

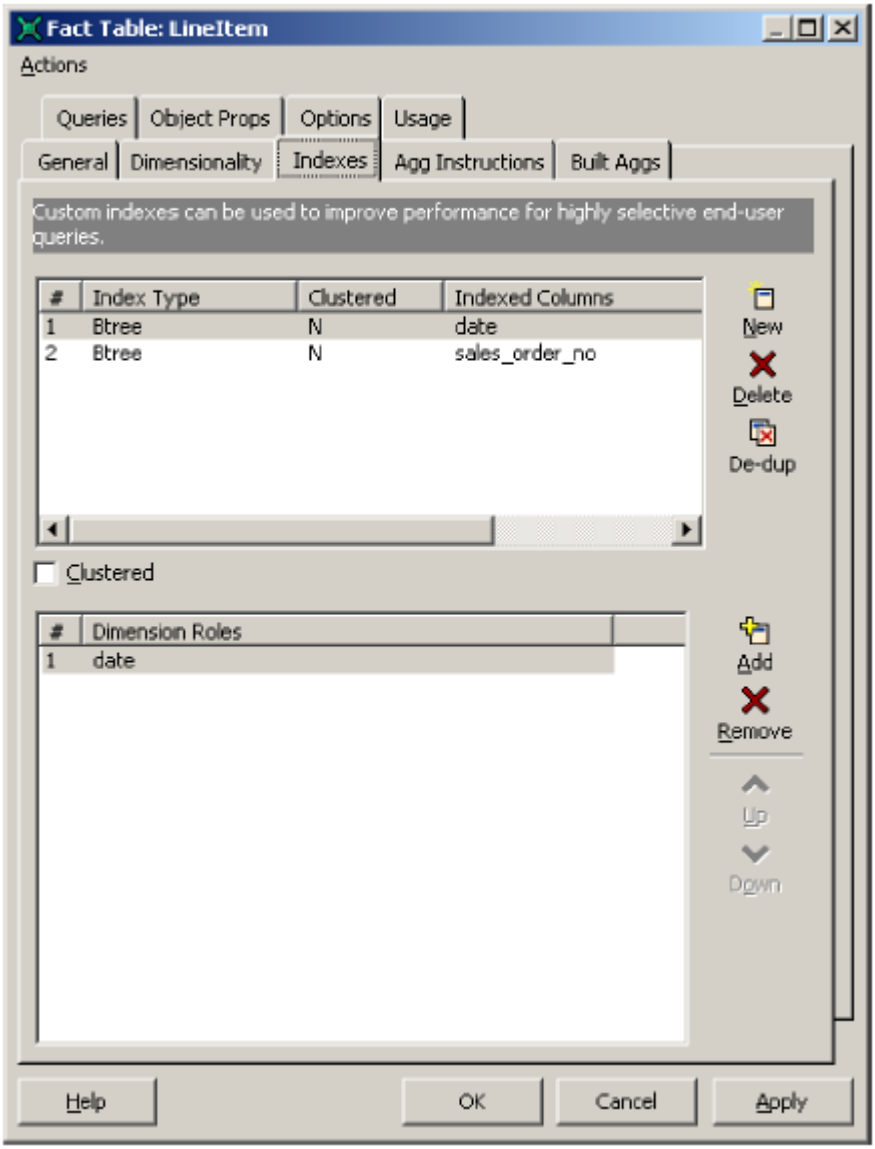


Figure 43: Fact Table Dialog Box: Indexes Tab

2 Select one or more dimension roles for this index from the Choose dialog box.

3 On an Oracle database server, select **Use Bitmap Index** if you wish to create a bitmap index.

Infor recommends that, if you are using an Oracle database server, you configure bitmap indexes for all dimensions with low cardinality. Bitmap indexes can significantly improve query response times because they are more space efficient and provide faster access.

4 Click **OK** to define the index. When you have defined the index, the dimension roles in the index are displayed in the Dimension Roles pane.

5 On a Microsoft SQL Server database server, you can specify one of the indexes as the clustered index. To specify an index as the clustered index, select the index in the upper pane and select the

Clustered option. Since a table can have only one clustered index, any index marked as **Clustered** reverts to **Btree** when you mark a different index as **Clustered**.

6 When an index is selected in the **Indexed Columns** pane, the dimension roles are displayed in the Dimension Role pane ordered by priority. The leading term of the index is at the top of the list. You can modify the dimension roles in an index as follows:

   a To change the sort priority of a dimension role, select that dimension role in the Dimension Roles pane and use **Up** and **Down** to move it to the desired location.

   b To choose additional dimension roles for the index, click **Add**.

   c To remove a dimension role from the index, select the dimension role and click **Remove**.

   d To delete an index, select the index in the Indexed Columns pane and click **Delete**.

   e To remove duplicate fact indexes (possibly after a schema merge), click **De-Dup**.

   Note: If your EpiMart database resides on a DB2 database server, the sum of the lengths of all columns in an index cannot exceed 1024 bytes. For example, you cannot define an index on five VARCHAR_255 columns.

Improving List and Campaign Query Performance

The query performance of transaction filters in lists and campaigns can be significantly improved by building an index on the fact table in which the leading column is the dimension role on which most lists and campaigns are based. All important fact columns and dimension roles must also be included as trailing columns in the index.

An index of this type allows grouping to be performed by stream aggregation rather than hash aggregation, which is slower. Note that any query involving dimension roles or fact columns that are not included in the covering index do not use stream aggregation.

Defining Fact Aggregates

The **Agg Instructions** tab of the Fact Table dialog box (see "Figure 44: Fact Table Dialog Box: Agg Instructions Tab" on page 129) allows you to define fact aggregates. As discussed in "Aggregate Building" on page 43, a fact aggregate is a substar of your star schema that uses aggregated versions of some of the available dimensions. The dimensions that you choose to use in the aggregate depend on which of the fact's dimensions end users can typically query as a group. You use Admin Manager to create aggregate instructions in metadata. The AggBuilder program uses these instructions to build the actual aggregates for the fact table.

Having the system aggregate fact values in advance speeds up front-end query time. In general, when designing fact aggregates, you must attempt to strike a balance between broad fact aggregates (which give low compression, but high query coverage) and narrow fact aggregates (which give high compression, but low coverage).

Note: If a fact table joins to an unmapped dimension, EpiChannel does not verify the mapping of the fact table to the dimension table when constructing the aggregate table. Therefore, if the fact table contains data which does not map to the dimension table, EpiChannel ignores it when constructing the fact aggregate table. This may cause aggregate reports to be incorrect.

Note the differences between unmapped and mapped dimensions:

- For an unmapped dimension, you must ensure that the corresponding dimension keys in any fact that joins to the dimension are actually present in the dimension table.
- For a mapped dimension, semantics automatically ensure that facts do not contain invalid dimension keys for that dimension.

To configure instructions for the fact table, select the **Agg Instructions** tab. (See "Figure 44: Fact Table Dialog Box: Agg Instructions Tab" on page 129 .)

Figure 5-21: Fact Table Dialog Box: Agg Instructions Tab

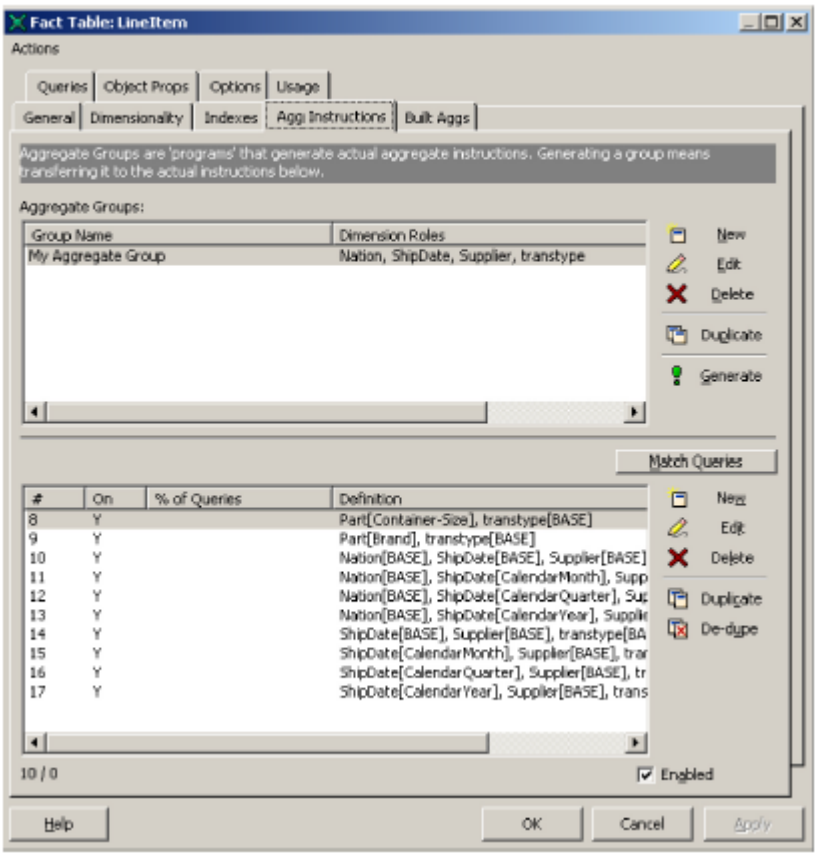


Figure 44: Fact Table Dialog Box: Agg Instructions Tab

Defining Aggregate Groups

Aggregate groups allow you to generate a collection of related aggregate instructions. The fact aggregates that can be generated from an aggregate group are determined by a combinatorial expansion based on the dimension role possibilities that you set up. See "Aggregate Building" on page 43 for additional information.

Creating an Aggregate Group

1 In the Aggregate Group area, click **New** .

2 In the **General** tab of the Aggregate Group dialog box (see "Figure 45: Aggregate Group Dialog Box" on page 130), enter the name of the group.

Figure 5-22: Aggregate Group Dialog Box

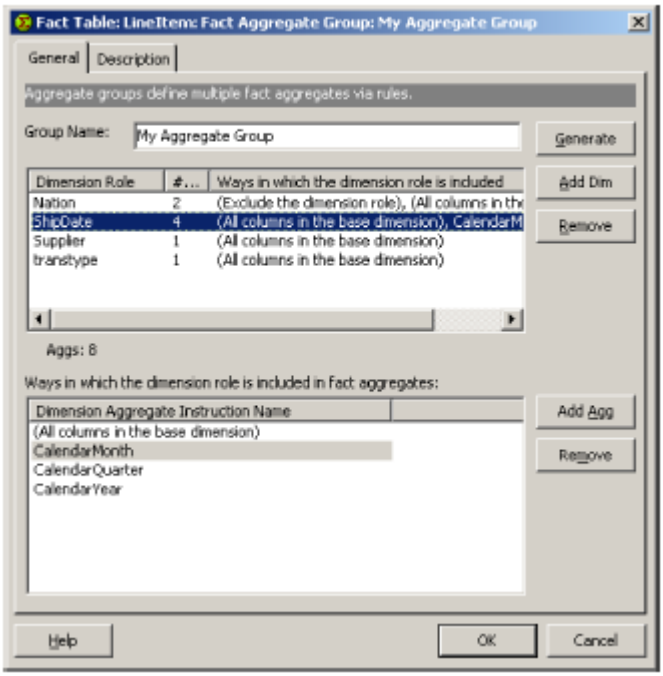


Figure 45: Aggregate Group Dialog Box

3 You can open the **Description** tab at any time to enter a description of this group.

4 Click **Add Dim** and select a dimension role for this group from the Choose dialog box.

5 Click **OK** in the Choose dialog box. The selected dimension role is displayed in the upper pane of the New Group dialog box and (All columns in the base dimension) is displayed in the lower pane. When you first add a dimension role to an aggregate group, the dimension role is included in its entirety. In addition, the `transtype` dimension role is added to every aggregate group.

6 You can change the ways in which the dimension role is to be included in fact aggregates. With the dimension role selected in the upper pane, you can insert the following changes:

   a Include the dimension role in more ways.

   Click **Add Agg** to display the Choose dialog box. This dialog box displays all aggregates that you have defined for the underlying base dimension of the selected dimension role, as well as the **(Exclude the Dimension Role)** option. Select the desired aggregates and options and click **OK** to add them.

   b Include the dimension role in fewer ways.

   You can select any aggregate or option in the lower pane and click **Remove** to remove it.

7 You can add additional dimension roles for the group and specify the ways in which you wish to include these dimension roles.

   The total number of aggregate instructions that are generated from the group is shown on the tab.

8 Click **OK** when you have finished defining the new group. The group is added to the listing in the upper pane of the Agg Instructions tab.

9 Create additional groups as described above. To add a group that is similar to an existing group, select the existing group and click the **Duplicate** button. Enter a name in the Duplicate dialog box, click **OK** , and then click **Edit** to modify the duplicate group.

Note the following:

- After you have defined the groups, you can generate the actual instructions in metadata by clicking **Generate** in the **Agg Instructions** tab. Generating creates aggregate instructions in the Actual Instructions pane.
- After you have added all desired instructions, you can click **De-Dup** to remove any duplicates that are created.
- After you defined an aggregate instruction, you can disable it without removing it from the Actual Instructions pane by selecting the instruction and deselecting **Enabled**.

Working with Aggregate Instructions

You can define an individual aggregate instruction by clicking the **New** button in the Instructions pane of the **Agg Instructions** tab. In the Aggregate Instruction dialog box ("Figure 46: Aggregate Instruction Dialog Box: General Tab" on page 132) that appears, specify the dimension aggregates to be used for this fact aggregate.

Defining and Editing Aggregate Instructions

1 Select a dimension role in the Fact Aggregate Specification pane.

2 Select a **Dimension Aggregate** for the dimension role from the drop-down list box. Select `BASE` to include the entire dimension. When you select a dimension aggregate, the columns in the aggregate are displayed under the dimension role name in the Fact Aggregate Specification pane and all remaining columns in the dimension are displayed in the Remaining Columns in Dimension pane.

Figure 5-23: Aggregate Instruction Dialog Box: General Tab

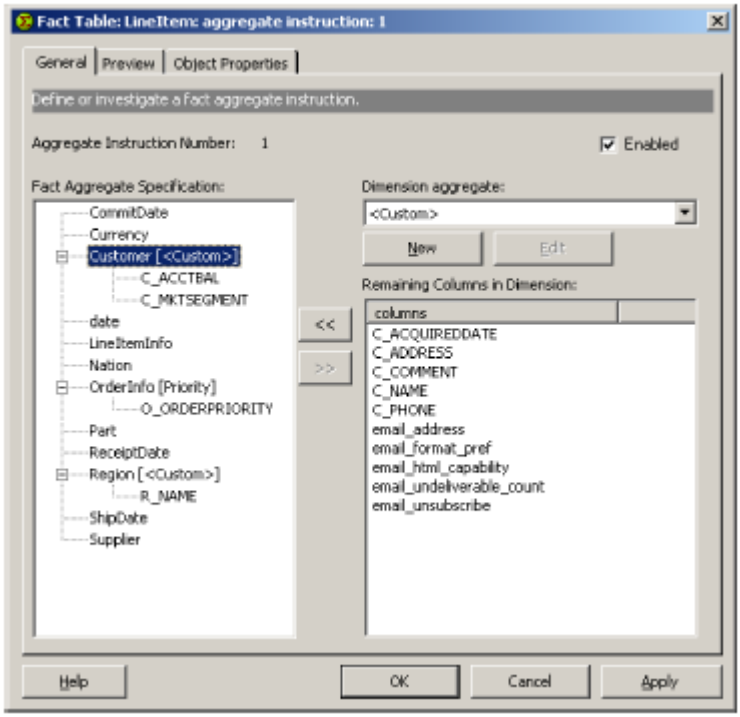


Figure 46: Aggregate Instruction Dialog Box: General Tab

3 If necessary, you can change a dimension-aggregate specification in this definition in the following ways:

   a To remove a dimension column from the dimension aggregate, select the column in the Fact Aggregate Specification pane and click the right arrow button.

   b To add a dimension column to the dimension aggregate, select the column in the Remaining Columns in Dimension pane and click the left arrow button.

   When you create these changes to dimension-aggregate specifications, the name of a dimension aggregate instruction with the selected columns is displayed in the **Dimension Aggregate** drop-down list box if such an aggregate instruction has been defined. If no such aggregate instruction has been defined, `<Custom>` is displayed. To define this custom dimension aggregate, click **New** to go to the Dimension Aggregate Instruction dialog box (see "Figure 33: Dimension Aggregate Dialog Box: Preview Tab" on page 110).

4 Follow the same steps to add any other dimension aggregates that you wish to include in this aggregate instruction.

5 You can preview the size of this proposed fact aggregate in the **Preview** tab. The Preview SQL pane shows the SQL that issues to determine the size of the aggregate. Click **Preview** to issue the SQL and display the size of the aggregate in the Preview Result pane.

6 You can change the physical object properties for this aggregate in the **Object Properties** tab. See "Physical Object Properties" on page 92, for a description of the properties that you can configure here and "Object Types and Views" on page 517 for a description of available object types.

7 Click **OK** to create the aggregate instruction.

You can edit any aggregate instruction that is shown in the **Instructions** tab of the Fact table dialog box. To edit an aggregate instruction, double-click it, or select it and click **Edit** , to display the Aggregate Instruction dialog box (see "Figure 46: Aggregate Instruction Dialog Box: General Tab" on page 132).

Removing or Disabling Aggregate Instructions

An aggregate group produce instructions that are infrequently used in actual user queries. Removing or disabling such instructions can significantly reduce extraction time and disk usage.

In general, if an aggregate instruction is no longer needed, you must disable it rather than removing it. If you remove an aggregate instruction that has been generated from an aggregate group, then that instruction is created again the next time that you generate aggregate instructions. If you disable an aggregate instruction, then that instruction remains disabled even after you generate aggregate instructions again.

To disable an aggregate instruction, select the instruction and deselect **Enabled**, or open the instruction's Aggregate Instruction dialog box and deselect **Enabled** in that dialog box.

Note: The aggregate instructions that you configure in the Fact dialog box are used the next time that the AggBuilder program is run. If a semantic instance has been applied to a fact table, then new aggregates are built in the Next partition. If no semantic instance has been applied, then new aggregates are built in the Current partition. See "Swapping Mirrored Tables" on page 192, for a discussion of the Next and Current partitions.

Optimizing Aggregates

Aggregates are best chosen based on actual query history. When your EpiCenter has accumulated a significant history of user queries, you must use query statistics (see "Fact Tables" on page 120) to fine-tune your aggregate definitions.

Browsing Fact Aggregates

As part of refining your EpiCenter, you can wish to browse the list of fact aggregates that AggBuilder built, in order to determine which tables are available for the query machinery. You can use the **Built Aggs** tab of the Fact Table dialog box (see "Figure 47: Fact Table Dialog Box: Built Aggs Tab" on page 134) for this purpose.

You can browse the A or B sets of aggregate tables as appropriate. Select the desired set of tables from the drop-down list under the **Filter** button. There are two copies of every fact table and fact aggregate, with suffixes `_A` and `_B` . The A and B tables are mirrored copies of the fact table and aggregates, and their operation is similar to that of the A and B dimension tables.

For every fact table, either the A or B tables are current at any time. If the current letter for a fact table is set to A, then all end-user queries that refer to that fact table are run against the versions of the base and aggregate tables for that fact that have a suffix of A. The B tables contain the data that was current before the most recent successful extraction, and they are inactive. See "Data Mart Mirroring: A and B Tables" on page 191 for a more details on table mirroring. You can choose to display only fact aggregates that include selected dimension aggregates.

Restricting Fact Aggregate Display

1 Select a dimension role in the Dimension Aggregate pane.

Figure 5-24: Fact Table Dialog Box: Built Aggs Tab

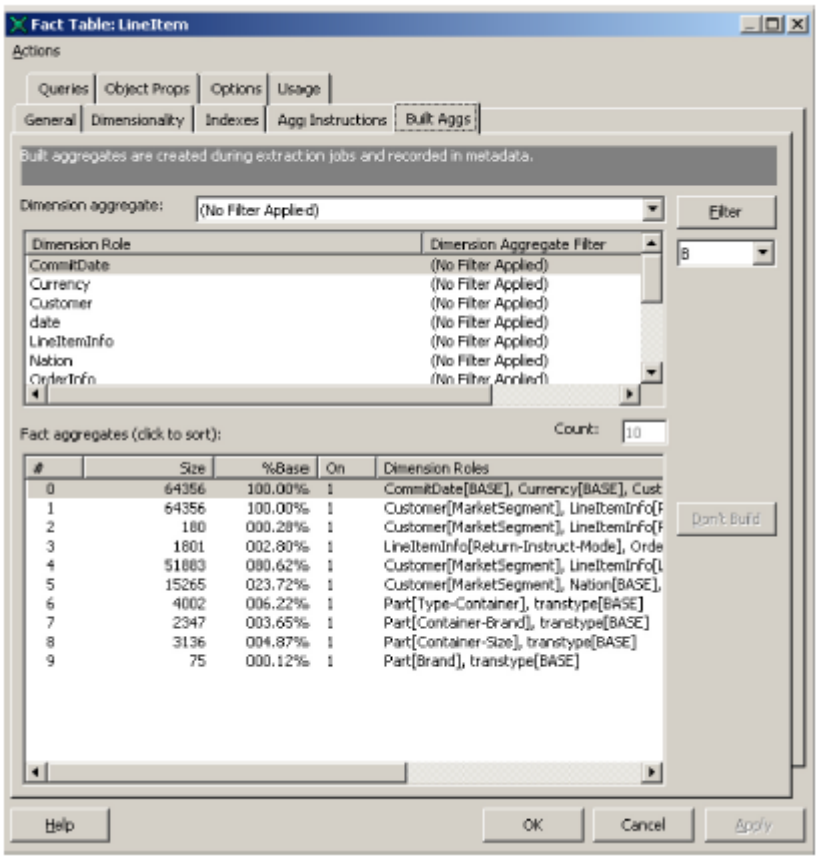


Figure 47: Fact Table Dialog Box: Built Aggs Tab

2 Select an aggregate for the selected dimension role from the drop-down list box. Select **(Not Included)** to list only aggregates that do not include the dimension role. Select **BASE** to list only aggregates that include the entire base dimension. Select **(No Filter Applied)** if you do not wish to filter based on this dimension role.

3 Click **Filter** to apply the selected filtering criterion to the display.

4 Repeat these steps for any other dimension roles that you wish to include in the display filter.

The Fact Aggregates pane displays the fact aggregates that match the filtering criteria that you have selected. Note that filtering only affects the display in this dialog box. The actual aggregates are not affected by the display filtering.

The Fact Aggregates pane shows the following information:

a The total number of fact aggregates built (with the current filter applied) is shown in the **Count** box.

b The fact aggregate number is shown in the # column.

c Although this numbering is arbitrary, it can be helpful when debugging query logs since the logs use these numbers.

d The `Size` and `%Base` columns show the size of the aggregate.

e The `Size` column shows the number of rows in the aggregate. The `%Base` column shows the size of the aggregate as a percentage of the size of the base fact table. You can click the `Size` or `%Base` heading to sort aggregates by size.

f The `Dimension Roles` column shows the dimension roles that are used in the aggregate.

The aggregate that is used is displayed in square brackets after the name of each dimension role.

Fact Query Statistics

Click the **Queries** tab of the Fact dialog box to display fact query statistics. (See "Figure 48: Fact Table Dialog Box: Queries Tab" on page 136.)

Note: Query statistics are only available if your EpiMeta and EpiOp databases reside on the same Oracle or SQL Server database server.

Query statistics are displayed in the upper pane of the dialog box. Each row shows statistics for a queried combination of dimension role columns on a fact table. By default, only the number of queries and the dimension roles in the query are shown. Click **Match Instructions** to show statistics about matching aggregate instructions.

Figure 5-25: Fact Table Dialog Box: Queries Tab

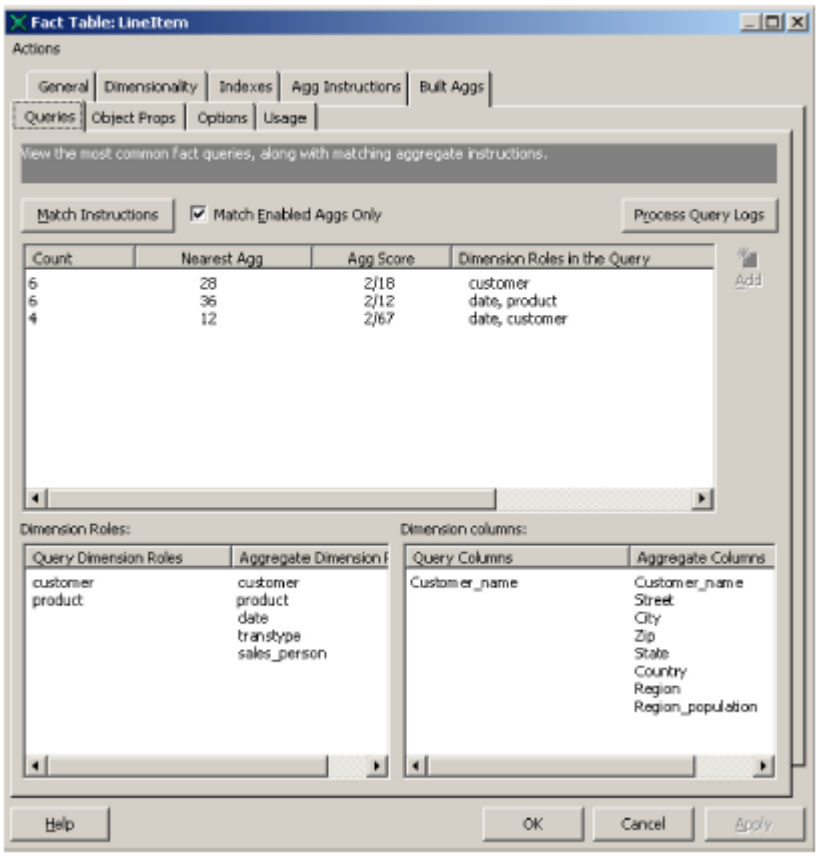


Figure 48: Fact Table Dialog Box: Queries Tab

The following statistics are displayed in each row:

- Count : The number of queries that can use of exactly this set of dimension role columns on this fact table.
- Nearest Agg : The fact aggregate, if any, that best matches the column set.
- Agg Score : The proportion of columns in the nearest aggregate, if any, that is used for this query. For example, if a query makes use of a total of three columns from two dimension roles and the nearest aggregate makes use of a total of eleven columns from four dimension roles, then the score is 3/11.
- Dimension Roles in the Query : The dimension roles that is used in the query.

Match Enabled Aggs Only is selected by default. Deselect it to include all aggregate definitions for consideration as possible matching aggregates. When it is selected, only enabled aggregates are matched to queries.

To view more details for a query, select the corresponding row in the upper pane. This displays the following information in the left pane at the bottom of the screen:

- The `Query Dimension Roles` column lists the queried dimension roles.

- If matching aggregate instructions are being displayed, then the `Aggregate Dimension Roles` column lists the dimension aggregates of the nearest matching fact aggregate, if any. Matching dimension aggregates are displayed adjacent to the matched dimension role.

If you select a dimension role in the left Query Details pane, the following information is displayed in the right Query Details pane:

- The `Query Columns` column shows the columns of the dimension role that is queried.
- If this query has a matching fact aggregate and matching aggregate instructions are being displayed, then the `Aggregate Columns` column shows the columns of the matching dimension aggregate in this fact aggregate. Matching aggregate columns are displayed adjacent to the matched query columns.

Click **Process Query Logs** to load or reload the query logs. When you click this button, the Process Query Logs dialog box (see "Figure 37: The Process Query Logs Dialog Box" on page 114) is displayed. In this dialog box, you can choose to reprocess all queries or to process only those queries that have not yet been processed. You can also set a query threshold. Queries issued less often than the queries threshold value are not processed.

To generate an aggregate instruction for a query, select the query and click **Add** to display the Aggregate Instruction dialog box (see "Figure 44: Fact Table Dialog Box: Agg Instructions Tab" on page 129).

Configuring Object Properties

You can configure vendor-specific physical object properties in the **Object Props** tab of the Fact dialog box (see "Figure 49: Fact Table Dialog Box: Object Props Tab" on page 138). See "Attributes and Granularity of Data" on page 33, for a description of the properties that can be configured here and "Object Types and Views" on page 517 for a description of available object types.

Any properties configured here override the object type defaults that are configured in the Physical Object Properties dialog box. See "The Alter Object Properties Utility" on page 391, for information on changing the properties of existing EpiMart objects.

Figure 5-26: Fact Table Dialog Box: Object Props Tab

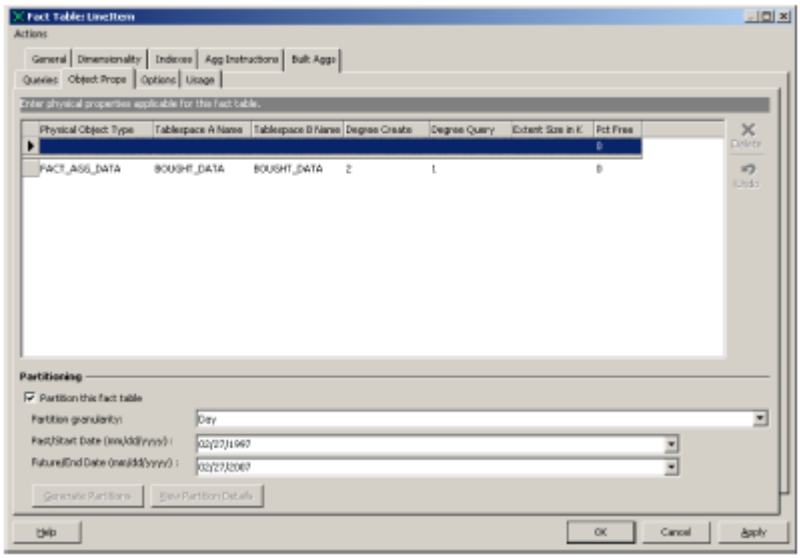


Figure 49: Fact Table Dialog Box: Object Props Tab

On an Oracle database server with the Oracle Partitioning option, you can also define fact partition properties in the **Object Properties** tab. Partitions are described in "Partitioning Oracle Base Dimensions" on page 36 and "Partitioning an Oracle Fact Table" on page 41. If a table is partitioned, then fact rolloff is performed by truncating all partitions that contain only data that is to be rolled off and deleting rows from partitions that contain both data that is to be rolled off and data that is not to be rolled off.

When you select **Fact Table is Partitioned**, you can set the following properties of fact-table partitions:

- **Partition Granularity**. The period of time for which data is included in each partition. Available options are **Day** , **Week** , **Month** , and **Year**.
- **Past/Start Date** . The earliest date for which you expect to extract fact data. This date must not be earlier than the earliest date in the Date dimension.
- **Future/End Date** . The latest date for which you expect to extract fact data. This date must not be later than the latest date in the Date dimension. If you attempt to extract data with a later date to this table, extraction fails.

Click **Regenerate Partitions** if you wish to recreate the partition instructions based on your date settings. If you regenerate partitions, any partition customizations details are lost.

Note: When you change partitioning instructions in this dialog box, including Regenerating Partitions, Admin Manager updates partition definition information in the metadata. Admin Manager makes no change to the EpiMart. To actually partition the tables in the EpiMart database, you must run Generate Schema.

Click **View Partition Details** to edit partition-level object properties and change the partition date ranges.

Configuring Fact Tables for Multiple Currencies

When you configure a fact table for transactions that can be denominated in different currencies, you have the following options:

1 You can display all facts to end users in a single standard currency, regardless of the currency that is used for the original transactions. Facts in the fact table are stored in the currency that is originally used for the transaction and are converted to the standard currency by measures. (See “Measures” in chapter 5 of the Topic Implementation Guide.)

   To implement this option, create a **Currency** dimension that contains a row for each type of currency that your organization uses when it conducts business. Include a **conversion** column that can be used to convert quantities in the original currency to a standard currency, such as the Euro. All measures that appear in Analytics web pages must use the **conversion** column as a multiplier to convert the original currency to the standard. Measure units (see "The Measure Units Tab" on page 149) control the way that the currency is formatted so that it corresponds to the correct currency.

   The **Currency** dimension can only include one row per currency, so Latest Dimension Value semantics must be used during extraction. Consequently, only the most current conversion rate is included in the **Currency** dimension.

2 You can display facts to end users in the currency of their choice. For example, Japanese users can view all transactions in Yen, and French users can view all transactions in Euros. In addition, this configuration allows you to compare values using the conversion rates that are current at the time of the transaction.

   To implement this option, create a **Conversion Rate** dimension in addition to the **Currency** dimension described above (the **Currency** dimension in this implementation does not require a **conversion** column). The **Conversion Rate** dimension must include conversion rates for all currencies into all other currencies. Facts in the fact table are stored in the currency that is originally used in the transaction and include a foreign key to the row in the **Conversion Rate** table that is current at the time of the original transaction.

   The Conversion Rate dimension can include multiple rows per currency, so Slowly Changing Dimension semantics can be used during extraction to maintain historical records of previous conversion rates. Consequently, users can compare values using the conversion rates that are current at the time of the transaction.

   Note: A measure cannot use a multi-currency measure unit unless a Currency dimension is defined in the EpiMeta database. This restriction is enforced by Scrutiny.

Creating a Currency Dimension

To create a currency dimension and indicate that a specific fact-table column can contain transaction values denominated in multiple currencies:

1 Double-click the **Measure Units** icon in the Configuration folder for your EpiCenter to display the **Measure Units** tab of the Configuration dialog box.

2 Choose a multi-currency measure unit. Do not enter a symbol for that unit. **Postfix** option must be deselected. **Multicurrency** option must be selected, and click **OK** . Note that the formatting options that you choose for this currency is the default value for all other currencies that do not have specific measure units.

3 Create a new base dimension for multiple currencies. Within that base dimension, include a column called SYMBOL. The SYMBOL column must be of a type, such as VARCHAR_5, that can hold a three-character currency name. This value must correspond with the value of the **Currency Name** option in the multi-currency measure unit.
4 Create a dimension role called CURRENCY that refers to your currency base dimension.
5 Add extraction job steps to populate the CURRENCY dimension with the symbol for every currency that appears in any multi-currency fact row.
6 Modify the extraction job steps for multi-currency fact tables to include the sskey of the currency dimension.
7 Create a measure that includes the column in which transactions in multiple currencies appear. Select the multi-currency measure unit that you just configured for this measure.

Displaying the Euro Character in a UTF-8 EpiCenter

The Euro character and other WS1252 characters that are not included in the ISO8859 Latin-1 code page can be stored incorrectly in an Oracle UTF-8 database. If you attempt to change the currency symbol to the Euro character in Admin Manager, question marks are displayed in place of the proper symbol, and web pages display an inverted question mark. To avoid this problem, perform the following steps.

1 On your Admin Manager machine, run regedit and add the following string-value registry key:

```
HKEY_LOCAL_MACHINE\Software\Epiphany\EpicenterManager\<version>\Form
Params\Preferences\NLS_LANG_OVERRIDE = <Language>_<Locale>.WE8MSWIN1252
```

where <Language>_<Locale> is any valid combination of language and locale. For example, you can use GERMAN_GERMANY, FRENCH_FRANCE, AMERICAN_AMERICA, and so on.

2 On your Infor Campaign Management Server machine, set the NLS_LANG Oracle parameter to the same value that is used on the Infor Campaign Management Platform machine. You can set this value in either the Oracle registry, or as an environment variable.
3 Reenter the Euro symbol in Admin Manager. It must display as the Euro character.
4 Set browser encodings to UTF-8 to display the symbol properly in end-user web pages.

Extracting Seed Data

Seeds are dummy records that are included in campaigns that target members of the dimension. They can be used to verify that campaigns are completed, or that fulfillment houses have not sold your campaign information to other vendors.

Note: Seed records are not included in the following:

- list/campaign counts
- communication tables (See "Built-in Fact Tables" on page 55 for details of communication tables.)

Setting a base dimension as list-producing and seed-using—specified in the **Options** tab of the Base Dimension dialog box—automatically generates an empty seed fact table for that dimension. EpiManager names the seed fact table dimensionName_shortID _sed, where dimensionName is the first 13 characters of the base dimension name, and shortID is the 2-character abbreviation for the dimension.

To populate the base dimension and the seed table, you need to run an extraction job and semantic for each of these two tables (base dimension table and seed fact table). "Figure 50: Customer Dimension, Seed and Source Tables" on page 141 illustrates the relationships between the base dimension table, the associated seed fact table and the initial source table (using the Customer dimension, and `my_seed_data_table` as the sample source of seed data).

Figure 5-27: Customer Dimension, Seed and Source Tables

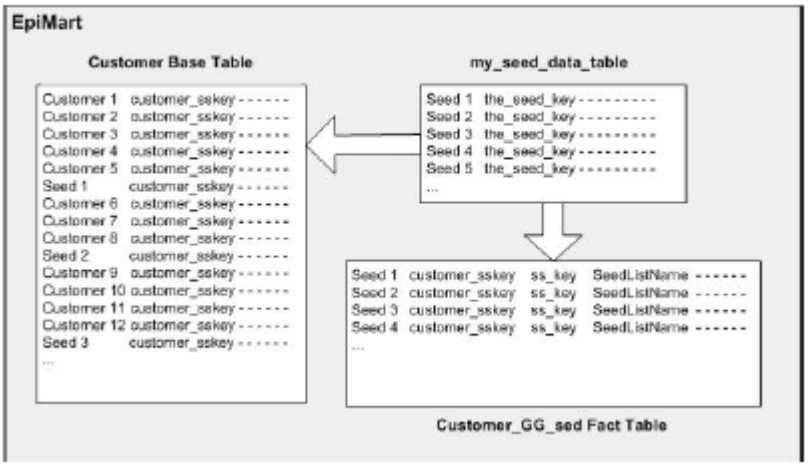


Figure 50: Customer Dimension, Seed and Source Tables

1 First, extract the seed data into the base dimension (Customer) table:

a Create a dimension extraction job step to extract the seed data from the temporary storage table. (The following SQL example uses `my_seed_data_table` as the data source.)

```
SELECT

the_seed_key customer_sskey,

$$TO_EPIDATE[ $$DBNOW ] date_modified,

$$NVL[ col1 ~,~ 'UNKNOWN' ] col1,

$$NVL[ col2 ~,~ 'UNKNOWN' ] col2,

$$NVL[ col3 ~,~ 'UNKNOWN' ] col3

FROM

my_seed_data_table
```

Note: Choose seed keys that do not conflict with actual data keys.

If there are only a small number of seed data rows, you can hard code the seed data directly into the SQL code, if desired.

2 Create a dimension semantic to move the seed data into the base dimension table (the Customer table in the example).

Figure 5-28: Example Seed Extraction Job

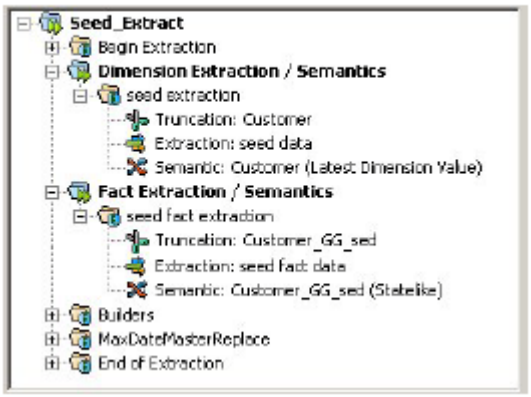


Figure 51: Example Seed Extraction Job

3 Extract the seed data into the seed fact table (`Customer_GG_sed`, in the example).

a Create a fact extraction job step to extract the seed data. (The following SQL example uses `my_seed_data_table` as the data source.)

Note: Infor recommends that you extract the seed data from the same source table in each step to ensure the integrity of the data.

```
SELECT
the_seed_key ss_key,
$$TO_EPIDATE[ $$DBNOW ] date_key,
1 transtype_key,
2 process_key,
$$NVL[ the_seed_key ~,~ 'UNKNOWN' ] customer_sskey,
$$NVL[ the_list_name ~,~ 'UNKNOWN' ] seedlistname_key,
1 occur
FROM
```

```
my_seed_data_table
```

b Create a fact semantic to move the seed data into the fact seed table (the `Customer_GG_sed` table in the example).

Relations

A relation defines a relationship between two list-producing base dimensions. For example, if you have a dimension that specifies information about employees, and a different dimension that specifies information about projects, you can define a relation that links the two dimensions together so that a user can:

- Create a query of the form, “Give me all Employees that are assigned to Projects with a certain property.”
- export a list of individuals that includes information from the Project dimension.

See "Relations" on page 49 for more information.

Defining a Relation

1 Right-click the Relations folder icon, and select **New Relation** to display the Relation dialog box (see "Figure 52: Relation Dialog Box: General Tab" on page 144). This dialog box has tabs labeled **General** and **Cardinality Labels**.

Figure 5-29: Relation Dialog Box: General Tab

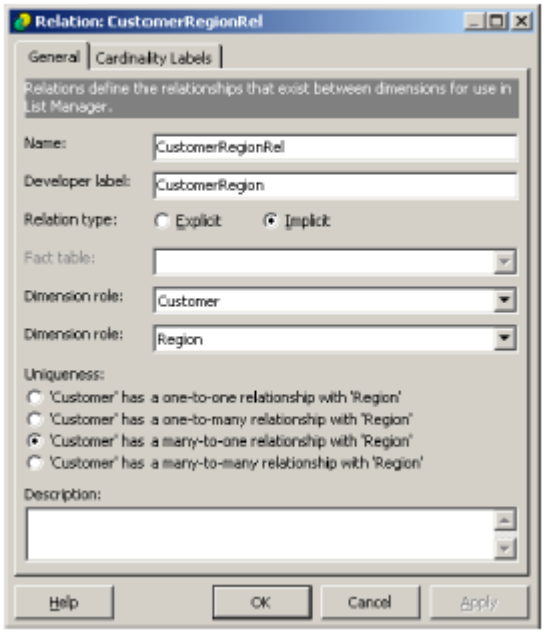


Figure 52: Relation Dialog Box: General Tab

2 Enter a name and developer label for the relation. **Name** is the name of the relation as defined in EpiMeta. **Developer Label** is the name of the relation as displayed in Admin Manager.

3 Specify whether this relation is explicit or implicit. See "Explicit and Implicit Relations" on page 51.

4 In the **Fact table** list box, specify a fact table that defines an explicit relationship between two dimension roles. For every combination of dimension role values, the fact table contains a row that indicates whether one dimension role value is related to the second dimension role value. See "Relation Fact Tables" on page 51 for information about required schema elements for this fact table.

   If you are defining an implicit relation leave this parameter blank.

5 In the two **Dimension role** list boxes specify two dimension roles that are related. For example, you can choose the `projects` dimension role in one list box and the `employee` dimension role in the other if you are defining a relation between projects and employees. You can specify the dimension roles in any order.

   Note: Both dimensions must be list-producing. A dimension is list-producing if the List is from this dimension option is selected in the Options tab of the Base Dimension dialog box.

6 Specify the type of relation that is being defined.

   - One-to-one : Each element of dimension A has a single unique match with an element of dimension B. For example, nations have a one-to-one relationship with capitols because every nation has only one capitol, and a capitol only exists in one nation.
   - One-to-many : Each element of dimension A can have multiple relationships with elements in dimension B, but each element in B has only one relationship with an element in A. For example, families have a one-to-many relationship with individuals because no individual can belong to more than one family, but multiple individuals can belong to the same family.

- Many-to-one : Each element of dimension B can have multiple relationships with elements in dimension A, but each element in A has only one relationship with an element in B. For example, individuals have a many-to-one relationship with families because multiple individuals can belong to the same family, but no individual can belong to more than one family.
- Many-to-many : Each element of dimension A can have one or more relationships with elements in dimension B, and vice versa. For example, individuals have a many-to-many relationship with favorite foods, because an individual can have more than one favorite food, and a food can be the favorite of more than one individual.

7 Enter an optional description of the relation for your reference.

8 In the **Cardinality Labels** tab, specify the terms that you wish to use to describe quantities of objects selected from each dimension role. This "table" on page 145 provides some sample cardinality labels for a sample dimension named Employees.

Table 19: Sample Cardinality Labels

Cardinality Label	Entries for the Employee Dimension
At least one	At least one employee
Everyone	All employees
Exactly one	Only one employee
No more than one	No more than one employee
No one	No employees

The Schema Observatory

The schema observatory (see "Figure 53: Schema Observatory" on page 146) allows you to view a graphical representation of your data mart schema. To open the schema observatory, double-click the **Schema Observatory** icon in the Schema folder for your EpiCenter.

Figure 5-30: Schema Observatory

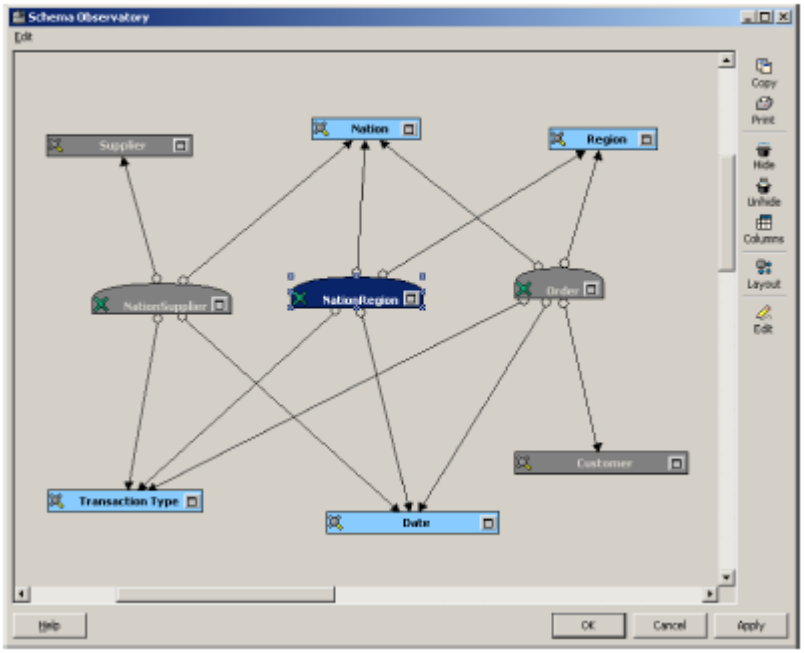


Figure 53: Schema Observatory

The schema observatory represents fact tables with rounded boxes and dimension roles with squared boxes. If a dimension role is included in the dimensionality of a fact table, then an arrow is shown going from the fact table to the dimension role.

When you select a table in the schema observatory, the table is highlighted in navy, and all tables to which it is connected are highlighted in light blue.

The table below "Table 20: Schema Observatory Operation" on page 146 lists the actions that you can take in the Schema Observatory.

Table 20: Schema Observatory Operation

Button or Menu Item	Action Description
CopyEdit > Copy	Copies the graph to the clipboard of your local machine.
PrintEdit > Print	Opens the Print Preview dialog box, from which you can print the graph.
HideEdit > Hide	Removes a selected node from the graph.
UnhideEdit > Unhide	Redisplays one or more nodes that you previously hid.
Columns(Or use the expand and shrink window icons that are displayed to the right of the table name.)	Displays or hides the columns in the selected table.
Layout	Redisplays the nodes in their original configuration.

Button or Menu Item	Action Description
EditEdit > Edit Selected Object(Or double-click anywhere in the fact table or dimension role boxes.)	Opens the selected fact table or dimension role for editing.
Edit > Select(Or use the CTRL key while selecting nodes.)	Selects more than one node at a time

The Configuration Dialog Box

The Configuration dialog box is the user interface for the `config_master` metadata table that contains various system parameters in name/value format. The information in this dialog box often requires little, if any modification. Many of these settings can be set using other Admin Manager dialog boxes, or specified automatically as you configure an EpiCenter.

With your EpiCenter selected, choose **Configuration** from the **EpiCenter** menu to view or modify these settings. You can go directly to any tab of the Configuration dialog box by double-clicking the appropriate icon in the Configuration folder of your EpiCenter.

The Configuration dialog box has the following tabs:

- **Settings**
- **Transaction Types**
- **Measure Units**
- **Strings**
- **EpiMart State**
- **User Preferences**
- **Date Formats**
- **Data Types**
- **Languages and Locales**

The sections that follow describe the tabs on which you configure data mart parameters. The **Strings** and **User Preferences** tabs are described in the *Topic Implementation Guide*.

The Settings Tab

The **Settings** tab of the Configuration dialog box in Admin Manager lists a number of settings that specify various optional behaviors of Infor Campaign Management components. These settings are classified by the types of behaviors they control. Each category of settings appears as a folder or subfolder in the Configuration dialog box ("Figure 54: Configuration Dialog Box: Settings Tab" on page 148). Configuration setting categories are defined in Admin Manager Online Help.

Figure 5-31: Configuration Dialog Box: Settings Tab

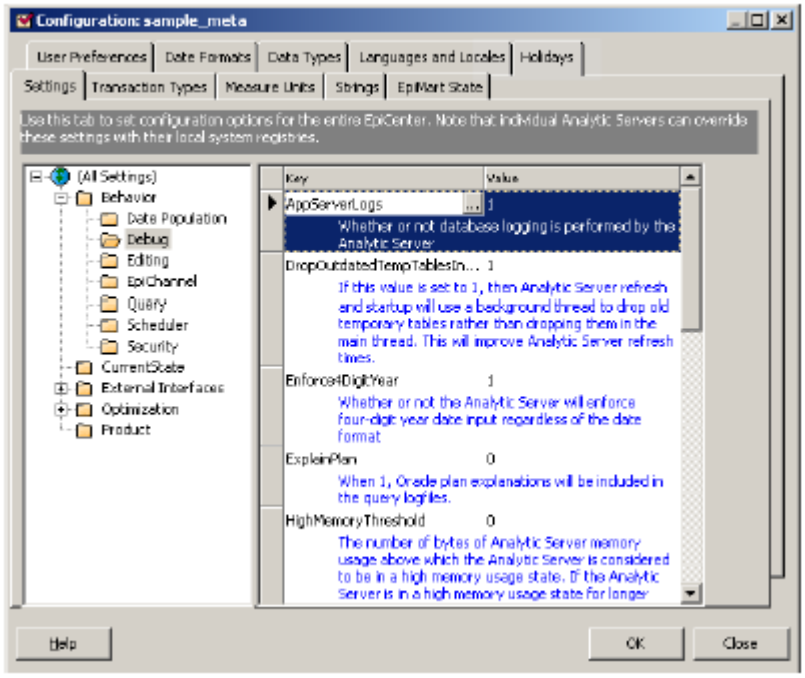


Figure 54: Configuration Dialog Box: Settings Tab

The configuration data that is displayed in the **Settings** tab of the Configuration dialog box (see "Figure 54: Configuration Dialog Box: Settings Tab" on page 148) is set for a standard EpiCenter. You need to create few, if any, changes to these settings.

If you need to modify these settings, take the following steps:

- Select the appropriate folder in the left pane.
- Select the key that you wish to change in the right pane.
- Enter a new value in the **Value** box. If there is a fixed number of values from which you can choose, **Value** is a drop-down list box from which you can choose the desired value. In other cases, **Value** is a text box.

  Settings are sorted into the following folders:

  - Behavior
  - CurrentState
  - External Interface
  - Optimization
  - Product

The Transaction Types Tab

The **Transaction Types** tab of the Configuration dialog box ("Figure 55: Configuration Dialog Box: Transaction Types Tab" on page 149) allows you to view the transaction types used by your EpiCenter, as well as customize additional transaction types for your site.

Figure 5-32: Configuration Dialog Box: Transaction Types Tab

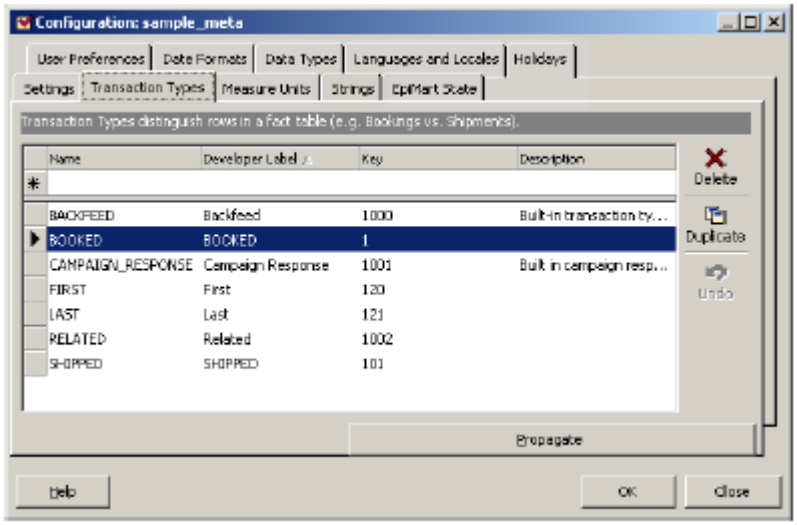


Figure 55: Configuration Dialog Box: Transaction Types Tab

Transaction type key values are limited to a maximum length of ten digits. Keys 0 -20,000 are reserved for use by Infor.

To add a new transaction type, enter **Name** , **Key** , and **Description** values in the top row of the dialog-box grid. The transaction type is automatically created when you press the `Enter` key or change focus from the grid row (for example, by selecting a different row, clicking a button, or going to a different tab).

To edit an already existing transaction type, select the transaction type and change the **Name** , **Key** , and **Description** fields to the desired values. In general, it is not recommended that you rename an existing transaction type, since this affects the import of metadata. Instead of renaming a transaction type, you can create a new transaction type that is identical except for the name, and then delete the old transaction type.

Click **Propagate** to synchronize the `transtype_0` table in the EpiMeta and EpiMart databases. For more information on transaction types, see “Transaction Types” in chapter 5 of the Topic Implementation Guide.

The Measure Units Tab

Measure units define how currency units, percentages, and numerical units are displayed by default on web Pages. To define a measure unit, select the top row of the grid and enter values for the

parameters specified in the following table. The system generates a hidden key for the unit that you have defined. Admin Manager uses this key when flagging a measure with a unit designation.

Figure 5-33: Configuration Dialog Box: Measure Units Tab

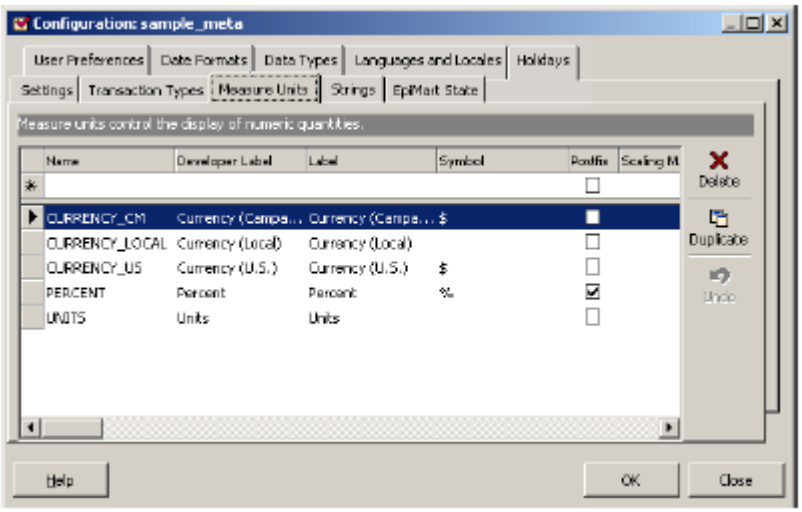


Figure 56: Configuration Dialog Box: Measure Units Tab

Parameter	Description
Name	The name of the measure unit as stored within your data mart.
Developer Label	The name of the measure unit as displayed in Admin Manager.
Label	The name of the measure unit as displayed to end-users.
Symbol	The symbol that must be used to demarcate the unit, such as $ or %. Symbols can be up to 50 characters long.
Postfix	Select this option if the symbol must follow the number: 100% is postfix, whereas $100 (the default) is prefix.
Scaling Magnitude	Specify whether you wish to scale the measure unit display for end users. For example, if you choose a Scaling Magnitude of 1000, the number 6000 is displayed as 6.
Number of Decimal Places	The number of digits to display to the right of the decimal marker.
Negative in Red	Select this option if you wish negative values to be displayed in red.

Parameter	Description
Negative in Parentheses	Select this option if you wish negative values to be displayed in parentheses.
Use Thousands Separator	Select this option if you wish to use a separating character between every third digit in large numbers. For example, with this option in place, 123456789 is displayed as 123,456,789.
Is Multi Currency	Select this option if your data mart uses currencies from different countries. Multi-currency controls how the system handles currencies. (A GROUP BY clause is forced on the currency dimension so that the system does not calculate different currencies, such as French francs and British pounds, as the same currency.) **Note:** A measure cannot use a multi-currency measure unit unless a Currency dimension is defined in the EpiMeta database. This restriction is enforced by Scrutiny. See "Fact Tables" on page 120, for more information on the use of multiple currencies in a fact table.
Parent Measure Unit	The measure unit that provides the default format settings for a multi-currency measure. If there is no measure unit for a particular currency, the currency inherits the format settings of its parent.
Currency Name	The name of the currency. This value must correspond to the value of the Symbol parameter in the Currency dimension of your EpiMart.
Description	An optional description of the measure unit for your reference.

The Strings Tab

Infor Campaign Management maintains a repository of predefined strings in metadata that the Infor Campaign Management Server uses to render certain labels and descriptive-text fields that appear in web pages. The **Strings** tab of the Configuration dialog box displays these strings ("Figure 57: Configuration Dialog Box: Strings Tab" on page 152).

Note: See “The Strings Repository” in the *Infor Campaign Management Topic Implementation Guide* for a description of the fields in this dialog box.

To add a new string to the string repository, enter information in the new-entry row of the **Strings** tab. This row is highlighted with an asterisk and appears at the top of the strings list.

Figure 5-34: Configuration Dialog Box: Strings Tab

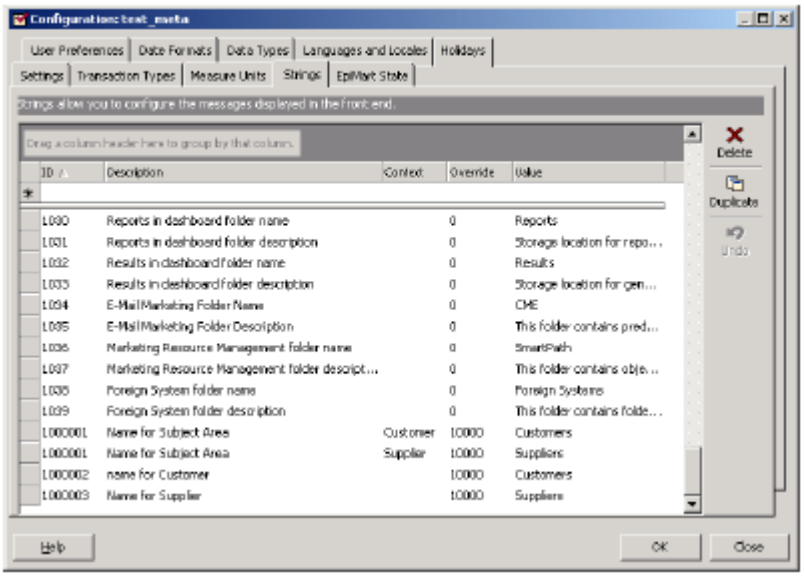


Figure 57: Configuration Dialog Box: Strings Tab

The EpiMart State Tab

The **EpiMart State** tab ("Figure 58: Configuration Dialog Box: EpiMart State Tab" on page 153) allows you to view the states of mirrored EpiMart fact and dimension tables (see "Data Mart Mirroring: A and B Tables" on page 191, for a discussion of mirrored EpiMart tables). For every fact, base dimension, and backfeed table, this tab lists the mirrored tables (A/B) that are in the Current state and the tables, if any, that are in the Next or Previous state. This tab also lists the state of backfeed tables in the EpiOp database. (See "Mirroring in Backfeed Tables" on page 191.)

Figure 5-35: Configuration Dialog Box: EpiMart State Tab

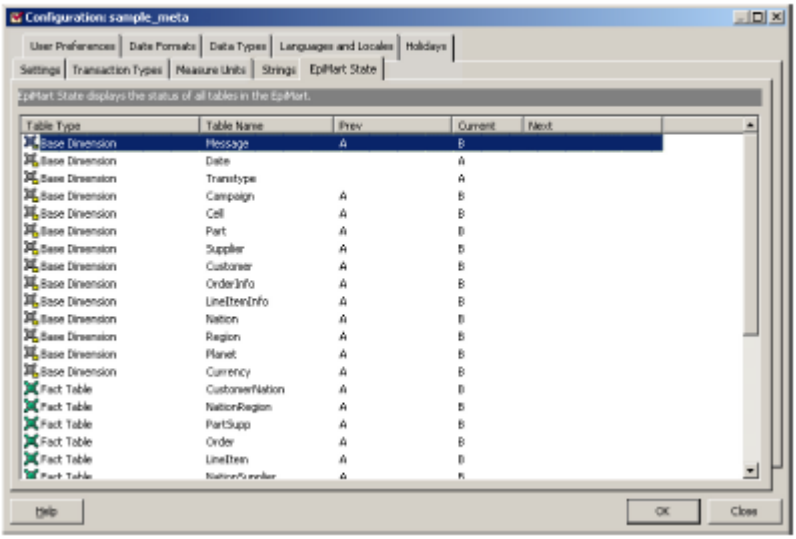


Figure 58: Configuration Dialog Box: EpiMart State Tab

The User Preferences Tab

The **User Preferences** tab of the Configuration dialog box allows you to set default user preferences for your Infor Campaign Management applications. This tab also allows you to specify which users can view and modify a particular user preference through the Web-based user interface (the General Preferences page) and which users cannot do so.

The **User Preferences** tab uses a grid control to give you editing access to the values of default user preferences ("Figure 59: Configuration Dialog Box—User Preferences" on page 154). To change the value of a user preference, click the **Value** field for that preference and then either edit the text field or select a value from the drop-down list, as applicable.

Note: See “Default User Preferences” in the *Infor Campaign Management Topic Implementation Guide* for a description of this dialog box and the order of precedence for override values.

Figure 5-36: Configuration Dialog Box—User Preferences

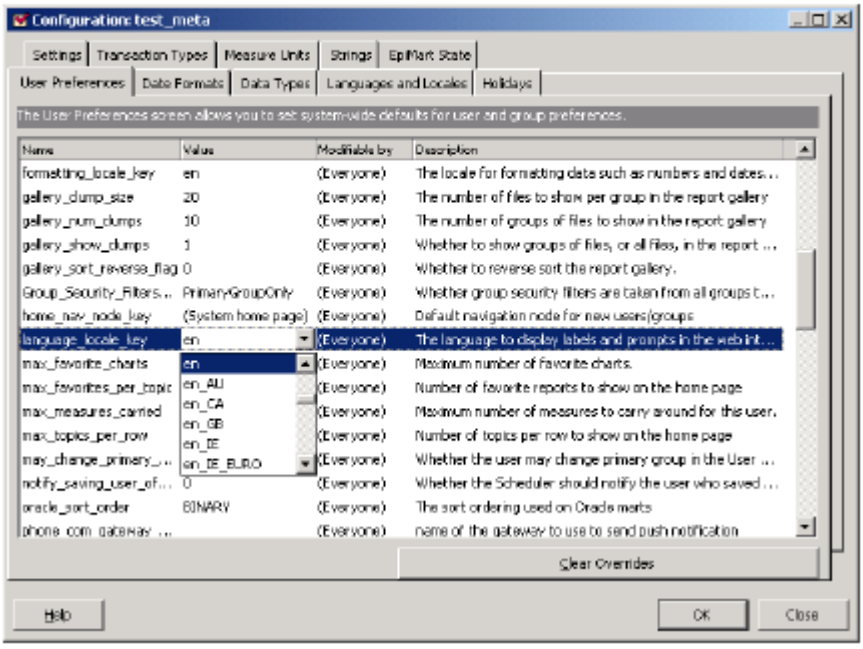


Figure 59: Configuration Dialog Box—User Preferences

The Date Formats Tab

You use the **Date Formats** tab to configure the appearance of dates in the end-user interface for different locales.

Figure 5-37: Configuration Dialog Box: Date Formats Tab

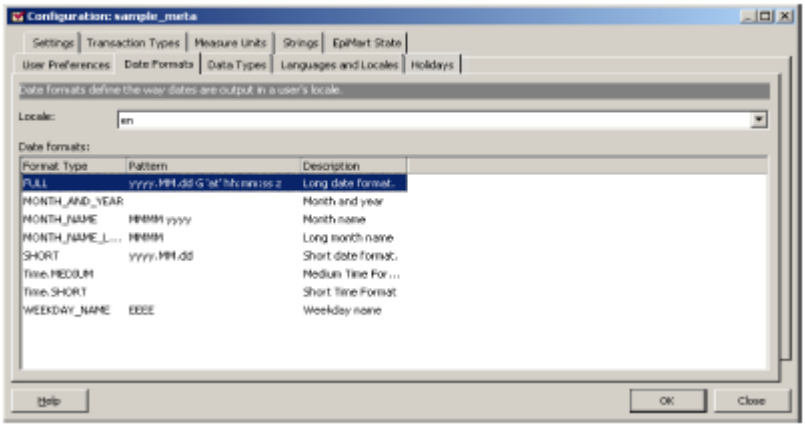


Figure 60: Configuration Dialog Box: Date Formats Tab

Note the following:

- Attributes of type DATE are formatted using the `SHORT` or `FULL` date formats.
- Special date dimension column attributes (`month_name`, `month_and_cy_name`, `month_and_fy_name`, `weekday`) are formatted using the `MONTH_NAME`, `MONTH_AND_YEAR`, `WEEKDAY_NAME` formats.
- The localized calendar uses `MONTH_AND_YEAR` , and `WEEKDAY_NAME` .
- Time formats are used for displaying time throughout the application, including the Task Manager, the current date in the header of each page, and so on.

These date formats correspond to Java built-in date formats. For each date format you can override Java's built-in pattern:

- In the Locale list box, select the locale for which you wish to configure your date formats.
- For each **Format Type** in the date format list, type in your own pattern based on the SimpleDataFormat Java class syntax (see "The Date Formats Tab" on page 154). The pattern you enter is validated by Admin Manager.

If the date format entry is left blank, the default is used. The defaults for these patterns are listed in Table "Table 21: Default Date Formats" on page 155 for the en_US locale.

Table 21: Default Date Formats

Date Format	Default Value
FULL	EEEE, MMMM d, yyyy
SHORT	M/d/yy
MONTH_NAME	MMM
MONTH_NAME_LONG	MMMM
MONTH_AND_YEAR	MMM yyyy
WEEKDAY	EEE
Time.SHORT	h:mm:ss
Time.MEDIUM	h:mm:ss a

Time Format Syntax

To specify a time format, use a time pattern string. In a time pattern string, all ASCII letters are reserved as pattern letters, which are defined in the following table.

The count of pattern letters determines the format:

- Text: Use 4 or more pattern letters for the full form of the word. Use less than 4 pattern letters for a short or abbreviated form if one exists.
- Number: The pattern letter count specifies the minimum number of digits. Shorter numbers are padded with zeros to this amount. Note that year is a special case: if the count of 'y' is 2, the year is truncated to 2 digits.
- Text and Number: Use 3 or more pattern letters for text. Use less than 3 pattern letters for a number.

Table 22: Date Format Syntax

Format Pattern	Result	Presentation	Example
G	era designator	Text	AD
y	year	Number	1996
M	month in year	Text or Number	July or 07
d	day in month	Number	24
h	hour in am/pm (1-12)	Number	12
H	house in day (0-23)	Number	21
m	minute in hour	Number	30
s	second in minute	Number	55
S	millisecond	Number	978
E	day in week	Text	Tuesday
D	day in year	Number	189
F	day of week in month	Number	2 (2nd Wed in July)
w	week in year	Number	45
W	week in month	Number	2
a	am/pm marker	Text	PM
k	hour in day (1-24)	Number	24
K	hour in am/pm (0-11)	Number	0
z	time zone	Text	Pacific Standard Time
‘	escape for text	Delimiter	
‘‘	single quote	Literal	‘

Any characters in the pattern that are not in the ranges of ['a'..'z'] and ['A'..'Z'] are treated as quoted text. For instance, characters like ':', '.', ' ', '#' and '@' appears in the resulting time text even if they are not placed within single quotes.

A pattern containing any invalid pattern letter results in a thrown exception during formatting or parsing.

Date Format Examples Using the US Locale

The following table displays examples of date format patterns in a US locale.

Table 23: Date Format Examples

Format Pattern	Result
"yyyy.MM.dd G 'at' hh:mm:ss z"	1996.07.10 AD at 15:08:56 PDT
"EEE, MMM d, "yy"	Wed, July 10, '96
"h:mm a"	12:08 PM
"hh 'o''clock' a, zzzz"	12 o'clock PM, Pacific Daylight Time
"K:mm a, z"	0:00 PM, PST
"yyyyy.MMMMM.dd GGG hh:mm aaa"	1996.July.10 AD 12:08 PM

The Data Types Tab

The **Data Types** tab ("Figure 61: Configuration Dialog Box: Data Types Tab" on page 157) allows you to define new data types. Data types define the size and type of data for fact, dimension, and external table columns.

Note: You can use the data type to set the column width of an attribute and thereby restrict the maximum length of data that a user can enter in the attribute entry field.

Figure 5-38: Configuration Dialog Box: Data Types Tab

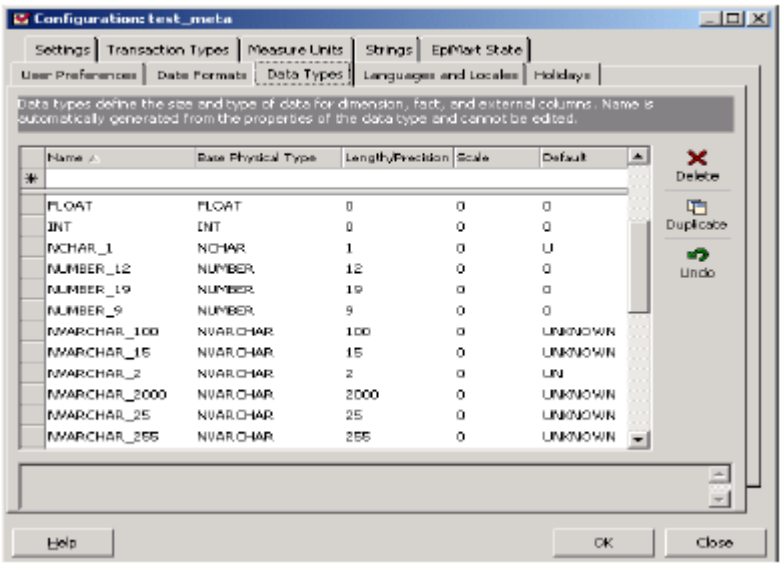


Figure 61: Configuration Dialog Box: Data Types Tab

The Existing Data Types pane displays all data types that are currently defined in your EpiCenter. If you wish to define a new data type, specify the parameters described in "Figure 61: Configuration Dialog Box: Data Types Tab" on page 157 and click **Add this data type**.

For additional information on data types, see "Data Type Values" on page 503

Note: NCHAR or NVARCHAR physical types allow you to store Unicode UCS-2 data on SQL Server databases. If you wish to include multiple language strings in your data mart, use these physical types. DB2 and Oracle use CHAR and VARCHAR physical types instead NCHAR and NVARCHAR.

Table 24: Configuration Dialog Box - Data Types Tab

Parameter	Description
Name	The name of your data type. This value is automatically generated based on the base physical type and length.
Base Physical Type	The type of data that is defined by your data type. When you select a value for this parameter, limitations for Length/precision and Scale are displayed in the text box below the value.
Length/precision	The number of characters or significant digits that can be stored by the data type.
Scale	The number of decimal places that must be stored for a data type that allows them.
Default	The default value for unknown values of the specified data type.

The Languages and Locales Tab

The **Languages and Locales** tab ("Figure 62: Configuration Dialog Box: Languages and Locales Tab" on page 159) allows you to specify which locales are enabled for your EpiCenter. For any locale, you can specify the options in the "Table 25: Configuration Dialog Box - Languages and Locales Tab" on page 159 table.

Figure 5-39: Configuration Dialog Box: Languages and Locales Tab

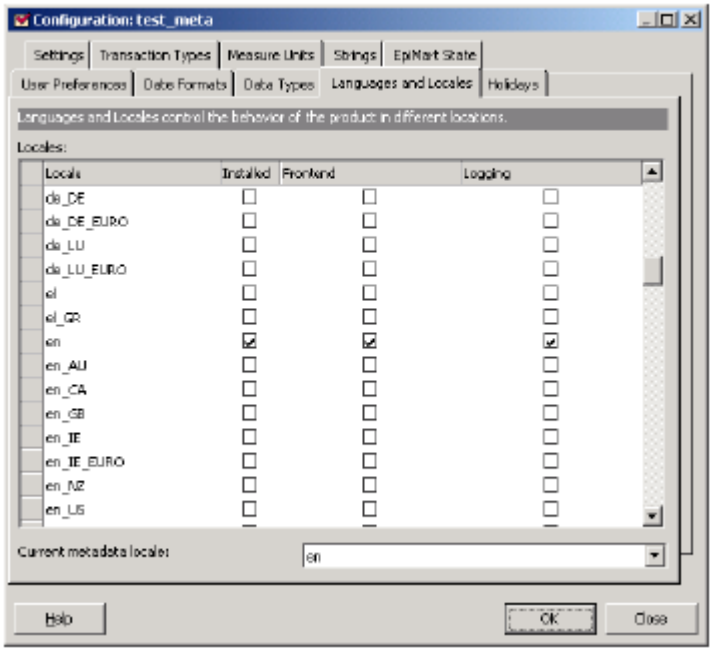


Figure 62: Configuration Dialog Box: Languages and Locales Tab

Table 25: Configuration Dialog Box - Languages and Locales Tab

Parameter	Description
Installed	If selected, Scrutiny generates warnings when objects are missing labels for this locale. If you wish to permanently delete all labels that are associated with a language, select the Tools menu and choose Internationalization > Deinstall Locale.
Frontend	If selected, this locale is available to end users in the Infor Campaign Management web interface.
Logging	If selected, this language is used for Scrutiny warnings and log files that are generated by EpiChannel. You can have only one logging locale at a time.

In the **Current metadata locale** list box, select the language that must be used for object Name fields in the EpiMeta for this EpiCenter. This language selection is displayed next to the EpiCenter name in Admin Manager's main window.

For more information on using multiple languages in your EpiCenter, see "Multi-Lingual EpiCenters" on page 78.

The Holidays Tab

The **Holidays** tab ("Figure 63: Configuration Dialog Box: Holidays Tab" on page 160) allows you to specify lists of holidays to be used when choosing not to run a campaign on holidays. When a user schedules a campaign, the user is given the option to skip holidays when running the campaign. If a user chooses to skip holidays, then the user must also select a holiday list, which determines the days that qualify as holidays for this campaign and the time zone that is used to determine the campaign execution date.

The Holidays tab shows the currently defined holiday lists in the Holiday Lists pane. A list of Standard US Holidays is included by default, and additional holiday lists can be defined as needed. When a holiday list is selected, the holidays in that list are shown in the Holidays pane.

Figure 5-40: Configuration Dialog Box: Holidays Tab

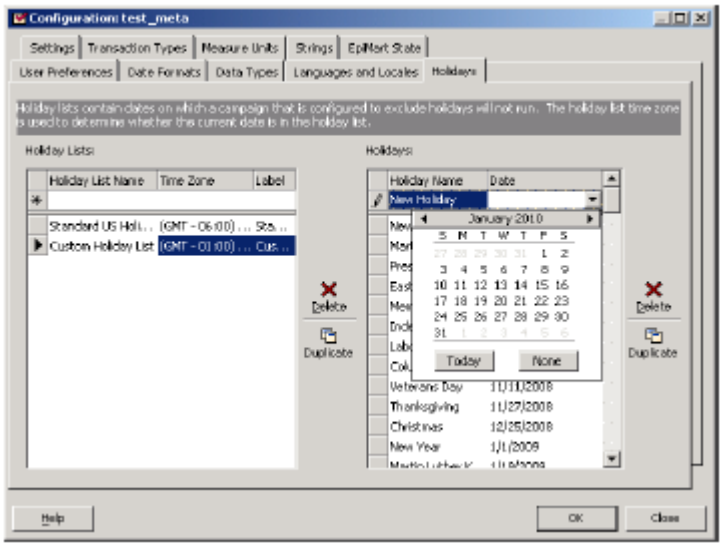


Figure 63: Configuration Dialog Box: Holidays Tab

To define a new holida list, enter the holiday list information in the top line of the Holiday Lists grid control. To edit a holiday list, select the holiday list in the Holiday Lists grid control and make any desired changes. The following table describes the parameters needed for a holiday list.

Table 26: Configuration Dialog Box - Holidays Tab, Holiday Lists field

Parameter	Description
Holiday List Name	The unique name of the holiday list.
Time Zone	The time zone to use with your holiday list. The scheduler skips campaign execution if the current date in the time zone associated with selected holiday list is in the list of holiday dates.
Label	A descriptive label for the holiday list. This label is shown when users select a holiday list to use with a campaign.

To add holidays to a holiday list, select the desired holiday list in the **Holiday Lists** grid control and then add new holidays on the first line of the **Holidays** grid control. To edit holidays in a holiday list, select the desired holiday list in the **Holiday Lists** grid control and then edit the holidays in the **Holidays** grid control. The following table describes the parameters needed for a holiday.

Table 27: Configuration Dialog Box - Holidays Tab, Holidays field

Parameter	Description
Holiday Name	A descriptive name for the holiday. This name need not be unique.
Date	The holiday date. This date is chosen using the drop-down calendar.

Generating the Schema

After defining your metadata via Admin Manager, you need to convert these definitions into actual tables in the EpiMart database. Before generating the schema, verify that your EpiMart data store refers to the database that you are using as your EpiMart database. Do this by opening the Data Store dialog box for the EpiMart data store. See "Defining a Data Store" on page 205 for more information about the data store dialog box.

You use the Generate Schema dialog box to build the EpiMart fact and dimension tables, to build external tables in the EpiMart database (for external tables that are configured to be generated), and to build the backfeed tables in the EpiOp database. You can also use the Generate Schema dialog box to populate the physical **Date** dimension table and to populate the sampling tables that are used by the campaign-management query engine.

Note: You cannot generate the schema if your EpiOp database has not yet been initialized. See "Initializing the EpiOp Database" on page 89 for more information.

Generating the EpiMart Schema

1 Choose **Generate Schema** from the **EpiCenter** menu to display the Generating EpiMart Schema dialog box. This dialog box has two tabs: **Schema** and **Results**.

Figure 5-41: Generating EpiMart Schema Dialog Box

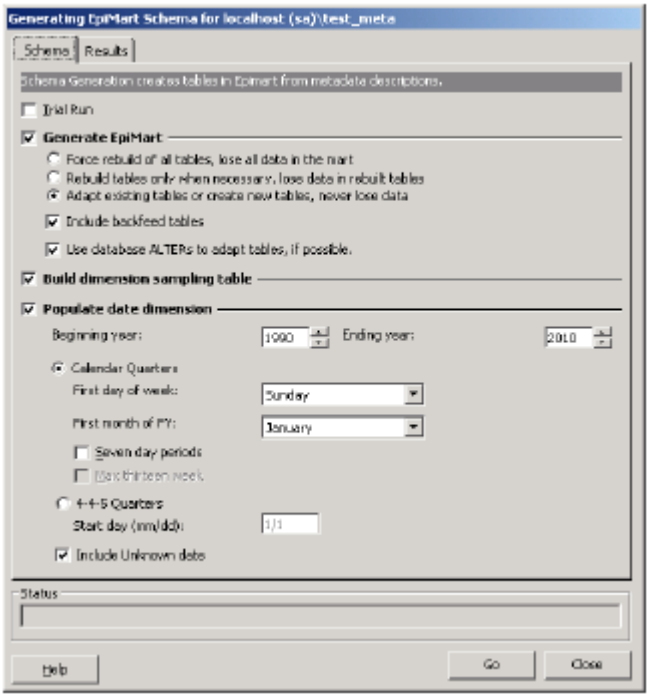


Figure 64: Generating EpiMart Schema Dialog Box

2 In the upper pane of the **Schema** tab, select **Trial Run** to see what the results are without making permanent changes to the files.

3 Select the **Generate EpiMart** option and select a generation method:

- Select **Force rebuild of all tables** only if you want to drop all existing data in the EpiMart database and completely rebuild all of your tables based on the current specifications in metadata.
- Note that you must use this option if you wish to change UNKNOWN values in your data mart for a particular dimension.
- Select **Rebuild tables only when necessary** to adapt tables to new schema specifications without losing data wherever possible. If tables cannot be adapted without losing data, then those tables are rebuilt and data in them is lost.
- Select **Adapt existing tables or create new tables** to adapt tables to new schema specifications without losing data wherever possible. If tables cannot be adapted without losing data, then schema generation halts with an error. See "Data Type Conversions" on page 504 for information about data type conversions in this schema generation mode.

Note: When you force rebuild of all EpiMart tables, all data in the EpiMart database is destroyed. You need to re-extract all of your data after forcing rebuild of tables. Use initial load semantics whenever you repopulate newly-rebuilt tables.

If you force the backfeed tables to be rebuilt, by selecting Include Backfeed Tables when Force Rebuild of All Tables is selected, all data about campaigns that have been run is deleted. If you do not back up this data prior to rebuilding the backfeed tables, then the data cannot be recovered.

Note: Scheduling campaigns to run while generating schema in adaptive mode to change the backfeed table columns leaves the EpiOp backfeed tables in an incorrect state and cause campaign

export to fail in the future. Failing to restart the scheduler after generating schema and the campaign queue is not set to refresh EpiMeta also leaves the EpiOp backfeed tables in an incorrect state and cause campaign export to fail in the future. Ensure that you shut down the scheduler when you generate the schema.

4 To generate the special backfeed tables for campaigns, select **Include Backfeed Tables**.

5 To use database ALTERs whenever possible when adapting existing tables, select **Use database ALTERs to adapt tables, if possible**. If this option is selected, schema generation database ALTER commands to adapt tables in place if the table changes are compatible with the ALTER command on your database platform. If this is option is not selected, schema generation recreates any tables that need to be adapted. Note that this option only affects schema generation performance, and both adaptation by table recreation and adaptation using ALTERs preserve all existing data. In most cases, selecting this option gives the best schema generation execution time, but in some cases faster execution can be obtained by deselecting this option for extensive changes to large tables.

6 If your application uses lists or campaigns, select **Build Dimension Sampling Table** the first time that you generate the schema. The sampling table is a list of random numbers that Infor Campaign Management applications use to produce samples of dimension tables.

7 If you have not yet populated the Date dimension, then you must populate the Date dimension table the first time that you generate the schema. See "Populating the Date Dimension" on page 166 for details.

Note: Repopulating the Date dimension causes related aggregates, indexes and MomentumBuilder tables to be rebuilt. You must run a builder job to repair the dropped aggregates and Momentum accelerator tables.

8 Click **Go** to start the trial run.

9 After the trial run has finished, the console window opens and displays information about the tables that are updated. Click the **Results** tab to view Date dimension population information, if available.

A console window opens immediately when you run generate schema, either in trial run mode or in non-trial-run mode. In either mode, the console window displays generate schema output, which looks similar to epichannel output because, with release 7.1, the schema is generated by epichannel. See "Generate Schema Summary Logs" on page 163.

Do not close the console window before the schema generation is completed. Closing the console window stops the generate schema.

When generate schema is completed, the dialog box displays the results of the date population phase of generate schema, in the **Results** tab. The date population phase of gen schema, is not generated by epichannel and is not displayed in the console.

10 If the results are acceptable, deselect **Trial Run**, and click **Go** to generate the schema in your EpiMart database.

Generate Schema Summary Logs

The following sample summary logs display the log information created by either a trial run or a non-trial run generate schema. The summary logs and the console window display the same information.

Trial Run Summary Log

Actual Run Summary Log

Populating the Date Dimension

To populate the date dimension:

1. Select the **Populate Date Dimension** option.
2. Enter the values for the beginning and ending years in the EpiMart database. The date range in the EpiMart database must be at least as large as any dates that are found in the data you are extracting, including dates found in dimension tables, such as `customer_birthdate`. These values are recorded in the `config_master` table and can be viewed in the **Settings** tab of the Configuration dialog box (see "The Settings Tab" on page 147).
3. Select **Include Unknown Date** to include an `UNKNOWN` date in your Date dimension. The `UNKNOWN` date is a special row in the Date dimension that contains a date that is not found in your data. By default, this date is set to a value of 01/01/1900, but you can change it with the `$$UNKNOWN_DATE` macro. Date dimension character columns use the physical type default for that column. Boolean numeric columns use 2 as the default, and 0 is the default for sequential numeric columns.

   In most cases, you must have meaningful date values for the date dimension role for all fact-table rows, but you can not always have valid or meaningful values for other dimension roles that refer to the Date dimension. In such a case, you can use the `UNKNOWN` date value.

   You can change the value that is used for the `UNKNOWN` date by editing the $$UNKNOWN_DATE macro in Admin Manager. The unknown date value must always take the following form:

   ```
   mm/dd/yyyy
   ```

   Replace mm with the two-digit month. Replace dd with the two-digit day. Replace yyyy with the four-digit year.

   If you change the value of the UNKNOWN date, you must perform a force rebuild of your data mart if you wish to propagate the change to existing rows.

   Note: If the value of the $$UNKNOWN_DATE macro is included in the legal date range for your EpiCenter, an error is generated.

   See "The UNKNOWN Dimension Row" on page 182, for more information on the use of the `UNKNOWN` row.
4. Choose the appropriate quarter system: Calendar Quarters (three month divisions of the year) or 4-4-5 Quarters. 4-4-5 Quarters represents the 13-week-per-quarter calendar in which the months in the quarter are defined as consisting of 4 weeks, 4 weeks, and 5 weeks.
5. For Calendar Quarters, select the **First Day of Week** and the **First Month of FY** (first month of fiscal year) from the drop-down list boxes.
6. For Calendar Quarters , select **Seven Day Periods** if you wish to have week boundaries occur at quarter boundaries. This ensures that all weeks in a quarter, except possibly the last week in an ordering, have exactly seven days. That is, for a field that counts up in number (such as `week_number_fq`), the first twelve weeks always have exactly seven days. The number of days in the thirteenth (and possibly fourteenth) week depends on the quarter and the setting of the **Max Thirteen Week Qtr** option. For fields that count down in number, such as `week_number_til_end_fq`, the earliest week in a quarter has a variable number of days, and all other weeks have exactly seven days.

For example, in a 92-day quarter, if the **Max Thirteen Week Qtr** option is not selected, seven day periods give you the following values:

	1	2	3	4	5	6	7	8	9	…	84	85	86	87	88	89	90	91	92
	1	1	1	1	1	1	1	2	2	…	12	13	13	13	13	13	13	13	14
	14	13	13	13	13	13	13	13	12	…	2	2	1	1	1	1	1	1	1

If **Seven Day Periods** is not selected, weeks can overlap quarter boundaries.

7 For Calendar Quarters , select **Max Thirteen Week Qtr** to avoid having a 14th week in a quarter or a 53rd week in a year. When this option is selected, the 13th week in a quarter is extended if necessary. Thus, for a 92-day quarter, Following values are shown in the table.

	1	2	3	4	5	6	7	8	9	…	84	85	86	87	88	89	90	91	92
	1	1	1	1	1	1	1	2	2	…	12	13	13	13	13	13	13	13	13
	13	13	13	13	13	13	13	13	12	…	2	2	1	1	1	1	1	1	1

8 For 4-4-5 Quarters , enter the start date of the first quarter.

For more information about date fields, see "Date Dimension Fields" on page 499 For a discussion of how the Infor Campaign Management treats time, see "The Date Dimension and Uniform Treatment of Time" on page 35.

After the schema has been generated, the system records the current state of the generated tables in metadata. The next time that the schema is updated, the current definitions can be compared with the new ones, and the appropriate tables can be created or altered as necessary.

Note: Whenever you change the structure of a fact table, a dimension table, or a generated external table in your EpiCenter, you must run the Generate Schema command in order to propagate those changes from your EpiMeta database to your EpiMart and EpiOp databases.

When you generate the schema, information about the Aggbuilder (see "AggBuild" on page 298) and MomentumBuilder (see "MomBuild" on page 299) accelerators is deleted for all rebuilt and adapted facts and dimensions. You must run extraction jobs that recreate these tables and indexes after you generate the schema. See "Rebuilding Tables After Schema Generation" on page 314, for more details.

Note: If you rename a table, do not create another table with the same name until you have run Generate Schema. This has the potential of failures and even data loss.

Rename the schema table in one run of Generate Schema, then recreate a new table with the original name in the next run of Generate Schema. That is:

- Rename original object.
- Run Generate Schema.
- Create the new object.

- Run Generate Schema again.

Database Extraction Overview

6

Note: Do not copy and paste scripts, commands or code from this document. Line breaks and some other characters picked up when you copy from the PDF can cause errors in the pasted text.

The goal of the extraction process is to copy raw data from your source systems, restructure it for use in your star schema, and add that data to your data mart. The extracted data is placed in your EpiMart tables, where it is accessible to front-end users that submit queries through Infor Campaign Management web pages. The extraction process involves the use of the following components:

- Admin Manager

  You use Admin Manager to define your extraction jobs and extraction steps, as described in "Extraction Steps" on page 269 and "Extraction Jobs" on page 303 You also use Admin Manager to define your star schema, as described in "Defining the EpiCenter Schema" on page 85

- Your EpiCenter

  An EpiCenter consists of EpiMeta , EpiOp, and EpiMart databases. The EpiMeta database contains the metadata that defines a star schema and the associated semantics. The EpiMeta database also contains data related to extraction jobs, as well as front-end configuration data. The EpiOp database contains operational data, such as campaign data, saved lists, logging tables, and backfeed tables.

  The EpiMart database consists of fact, dimension, external, and staging tables. The fact and base dimension tables hold the data for the star schema. The staging tables are used to populate these fact and dimension tables. The backfeed tables are used to store data about campaigns that run. External tables are additional tables that are available for use by the extraction machinery and accelerators.

- Data stores

  Data stores are data sources and destinations for an EpiCenter. Data stores include source databases, EpiMeta, EpiMart, and EpiOp databases, and logging directories.

- Semantic types, templates, and instances

  At a high level, a semantic type reflects the meaning of a company's business processes. During data extraction, semantic instances, which are compiled SQL programs, are used to merge newly extracted data into data loaded during previous extractions in a way that reflects these business processes. See "Semantics," on page 223 for definitions of terms related to semantics.

- EpiChannel

  EpiChannel is the Infor Campaign Management extraction program that executes jobs and logs job activity. See "Running Jobs with EpiChannel" on page 351 for more information.

- The Infor Campaign Management Scheduler

  You use the Infor Campaign Management Scheduler to schedule extraction jobs for automatic execution at specified times.

- Database administration tools

  Database administration tools, which are supplied by your database vendor, are used to administer the physical characteristics of the databases and the interconnections among these databases. Please refer to your database vendor's documentation for information about these tools.

Extraction Phases: An Overview

Infor Campaign Management database extraction is done using a "load and merge" model. First, source-system data is loaded into staging tables in the EpiMart database. After data has been loaded into the staging tables, that data is then merged into the main EpiMart fact and dimension tables.

Note: In some situations it is possible to bypass the staging tables and load data directly into fact and dimension tables. See "Streaming Extraction" on page 172, for more information).

The extraction process includes the following phases:

- Preparing the EpiCenter for extraction.
- Loading fact and dimension data into staging tables (pull/push)
- Applying semantic rules to merge staging-table data into the data mart
- Building aggregates to improve response times for user queries
- Building list and campaign accelerators
- Completing the extraction process by recording the extraction date, toggling the mirrored data mart tables, and building views for external reporting

The sections that follow describe these phases.

The extraction machinery makes use of staging tables in the EpiMart database and, optionally, external tables.

A staging table is a table that contains all of the fields required by a single base dimension or fact table. During the load, or pull/push extraction phase, data is extracted from a source database or file in raw format and loaded into data mart staging tables. Staging tables hold the data in preparation for the second phase of extraction: data merging. All required staging tables are automatically created when you generate your EpiCenter schema.

An external table is a table, the data stored here is available to Infor Campaign Management. The most common use for external tables is as extraction source tables. External tables can also serve as intermediate tables when more complicated multi-staged extraction is required. When external tables are used as intermediate tables for extraction, they are usually built in the EpiMart database and are thus internal to the EpiCenter.

Beginning of Extraction

By default, every EpiCenter includes a group of extraction steps that are designed to be run at the beginning of an extraction job. This group contains a step that cancels any other non-backfeed extraction jobs that are running or not have run to completion.

Data Loading

The data-loading phase of extraction copies data from your source systems into EpiMart staging tables (or, in the case of streaming extraction, directly to base dimension or fact tables). A staging table is an EpiMart table that has the same schema as its target EpiMart fact or dimension table.

The Infor Campaign Management Adaptive Schema Generator generates the staging tables at the same time that it generates the target tables, ensuring that these tables have matching structures. The Adaptive Schema Generator is the component of Infor Campaign Management that builds the tables in the EpiMart database using the schema metadata definitions in the EpiMeta database. See "Generating the Schema" on page 161 for more information on schema generation.

The Adaptive Schema Generator activities are recorded in the job log folder Generate_Schema, using the standard job log files:

- epiChannel_summary.log
- epiChannel_detailed.log

For more information on where these log files are stored, see "Log Directories and Files" on page 374.

When you run the EpiChannel program to start the extraction process, EpiChannel reads the metadata for the job that you have selected and begins executing extraction steps that copy data from an input data store to an output data store. In a relatively simple extraction job, data copied directly from a source system to the data mart staging tables. A more complicated extraction job can have steps that copy data from source systems into prestaging tables. The data in these prestaging tables can then be transformed and loaded into the staging tables by additional extraction steps.

At the beginning of an extraction job, staging tables and external tables contain data from a previous extraction job, so they must be truncated. In most extraction jobs, for every staging table or external table that is being populated, a truncation step is placed immediately preceding the extraction steps that populate that table.

At the end of the data-loading phase, all extracted data that is to be added to your EpiMart must be in the appropriate staging tables.

Data Merging

After the staging tables are fully loaded, semantic instances merge the new data from the staging tables into the EpiCenter tables.

When you configure an extraction job, you assign a semantic type to each fact or dimension table for which you are extracting data. This semantic type determines how the data from the staging table is to be merged into the data mart table.

A semantic type is a generic merging procedure that resolves potential conflicts within the data. Each semantic type is implemented as a generic SQL program called a semantic template that you can apply to your particular data mart tables. A semantic instance is an instance of such a semantic template is adapted for the specific fact or dimension table that is being populated. The Infor Campaign Management extraction program, EpiChannel, executes the semantic instance at the appropriate time during an extraction job.

Semantic instances allow you to specify how data updates are to be handled by Infor Campaign Management. For example, if a customer moves to a new location, you can choose to update all data with the customer's new location. If you, want to keep track of the changes in customer data over time. You can use the Slowly Changing Dimensions semantic type to track these changes. When you use this semantic type for the customer dimension, all fact data is automatically associated with the customer data that is current at the time that the fact occurred. For example, an order is associated with the customer's address at the time of the order.

Infor Campaign Management provide a large collection of built-in semantic types, which are described in detail in "Semantics" on page 223. You can also define additional semantic types, as described in "Defining New Semantic Types" on page 266.

Streaming Extraction

Using streaming extraction , you can bypass the staging tables and populate data mart tables directly. Streaming extraction can improve extraction performance, but since it bypasses the special logic of Infor's semantic instances, it places additional demands on job configuration and source-system data consistency.

For dimensions, only the initial load can be streamed. For fact tables, you can stream both the initial load, and any incremental loads.

Note: Streaming loads do not apply any de-duplication or data cleansing steps. Exercise caution when using this job type for incremental loads.

If you are using streaming extraction to populate or update a fact table, keep the following considerations in mind:

- All dimensions that are included in the dimensionality of the fact table must be unmapped or must be sufficiently small. If a mapped dimension that is included in the dimensionality of the fact table has more rows than the value set in the `DimmapMaxValuesWarn` configuration key, then a warning is issued. If a dimension has more rows than the value set in the `DimmapMaxValues` configuration key, then the job halts with an error.
- The streaming fact extraction step must be executed after the dimension semantics for all dimensions included in the dimensionality of the fact table. If you are extracting data from a changing source system, use time-based extraction filters to avoid data inconsistencies.

Backfeed Merging and Extraction

Backfeed tables store information about campaigns that are running. Campaign information includes lists, list segments, treatments applied, and responses. Whenever a scheduled campaign is run, the information for that campaign is automatically stored in the backfeed tables. The Backfeed Dimension and Backfeed Fact semantics are used during extraction to update the main backfeed tables with data from recently run campaigns.

This backfeed data is available to EpiCenter users by extracting it into the data mart tables. As with staging tables, backfeed tables have a schema similar to that of the target EpiMart tables. The Infor Campaign Management Adaptive Schema Generator generates the backfeed tables at the same time that it generates the target tables, ensuring that these tables have matching structures.

The Backfeed Dimension and Backfeed Fact semantics update only the data in the backfeed tables. In order to use this data in your data mart, you need to extract it from the backfeed tables. This extraction completes with the same load and merge process as extraction from other data sources. Data is copied from the backfeed tables to the EpiMart campaign staging tables by means of the usual extraction mechanism. Backfeed extraction is described in detail in "The Backfeed Job" on page 321. Data is moved from the campaign staging tables to the main EpiMart campaign tables using the same semantics that are used for other dimension and fact tables.

Building Aggregates

The next extraction phase is aggregate building. Aggregates are pre-calculated and pre-stored summaries of data that significantly accelerate the front-end query machinery. The query machinery is the component of the Infor Campaign Management Server that communicates with the EpiMart database and issues SQL statements against the RDBMS.

You use Admin Manager to define which facts are aggregated for groups of dimensions. The AggBuilder extraction step, which typically runs immediately following data extraction, performs roll-ups, or aggregations, on the groups and includes them as aggregates in the EpiMart tables. Aggregates are discussed in detail in "Aggregate Building" on page 43.

Building Indexes and Campaign Accelerators

The MomentumBuilder extraction step, which is normally run after aggregate building in an extraction job, builds indexes and creates and updates the special accelerators used to manage lists and campaigns. See "Star-Schema Extensions for Infor Campaign Management" on page 54, and "MomBuild" on page 299 for details.

End of Extraction

By default, every EpiCenter includes a group of extraction steps that are designed to be run at the end of an extraction job. This group contains steps that record the date of extraction in metadata, a step that toggles the current data mart tables, a step that creates data mart views for external reporting, and a step that refreshes cached metadata information in the Infor Campaign Management Server.

Toggling Data Mart Tables

In order to insure data integrity, the EpiMart database contains mirrored copies of all fact and dimension tables (see "Data Mart Mirroring: A and B Tables" on page 191, for details). Every dimension and fact table, as well as every fact history table, has a current and an inactive copy. End-user queries are always run against the current tables, and data is extracted into the inactive tables.

At the conclusion of extraction, the extracted data resides in the proper columns of the inactive tables in the EpiMart database. The data extracted from the source data store and placed in the EpiMart data store on the RDBMS server that holds your EpiCenter. When the data is successfully extracted in this way, the inactive tables must be set to be the current tables in order to use the new data available for end-user queries.

When the current tables are toggled, all inactive fact and dimension tables that have data extracted into them during this extraction jobs are current. When these inactive tables are current, the corresponding mirror tables that are current at the beginning of extraction are inactive.

Creating Views for External Reporting

An EpiCenter allows queries against the current contents of EpiMart tables by providing views for external reporting (see "Views for External Reporting" on page 197). In the End of Extraction group, these views are updated immediately after the data mart table states are toggled. Since these views have fixed names, they allow you to issue queries against the currently active data mart tables without being concerned with which of the mirrored tables are currently active.

Running Extraction

EpiChannel can execute an extraction job in standalone mode or in client/server mode. In standalone mode, EpiChannel runs locally on the host from which it is invoked, generally the host on which the Scheduler and Admin Manager is installed. In client/server mode, the local host invokes EpiChannel execution on a remote host, generally the host on which your EpiMart database resides.

Standalone Mode

In standalone mode, EpiChannel fetches data to the local host and then writes data to the database server. This process is illustrated in "Figure 65: Typical Standalone EpiChannel Execution" on page 175. EpiChannel can be invoked in standalone mode with either of the following methods:

- Through the command line, as described in "Running Jobs with EpiChannel" on page 351 When you run the `epichannel` command without the `-remote` option, EpiChannel executes locally.
- Through the Execute Job dialog box in Admin Manager0. To run the job locally, select the **Start in standalone mode** option.

Figure 6-1: Typical Standalone EpiChannel Execution

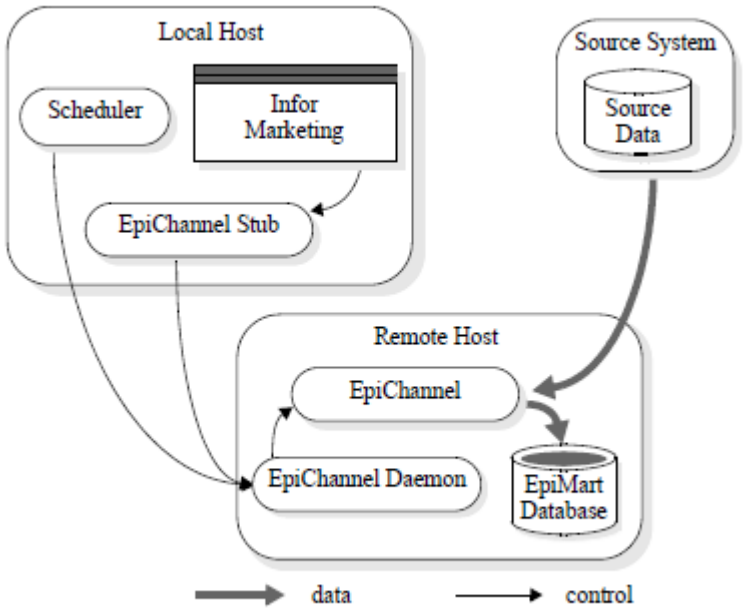


Figure 65: Typical Standalone EpiChannel Execution

Client-Server Mode

In client/server mode, EpiChannel typically runs on the host on which the database server for your EpiMart database resides. EpiChannel fetches data to the remote host and then writes that data to the database server on the same host. This process is illustrated in "Figure 66: Typical Client/Server EpiChannel Execution" on page 176.

EpiChannel can be invoked in client/server mode with any of the following methods:

- Through the command line, as described in "Running Jobs with EpiChannel" on page 351 Run the `epichannel` command with the `-remote` option to specify the remote host.

- Through the Execute Job dialog box in Admin Manager. To run the job remotely, select the **Start in client mode** option.
- Through the Scheduler. When the Scheduler determines that it is time for an extraction job to execute, it sends a message to the EpiChannel daemon to execute EpiChannel with the parameters specified in the Task Schedule dialog box.

When EpiChannel is invoked in client/server mode with either the command line or Admin Manager, an EpiChannel stub is spawned on the local host. This stub then passes the appropriate parameters to the EpiChannel daemon on the remote host, which spawns an appropriate EpiChannel instance to perform the extraction. The Scheduler passes parameters directly to the daemon, bypassing the stub.

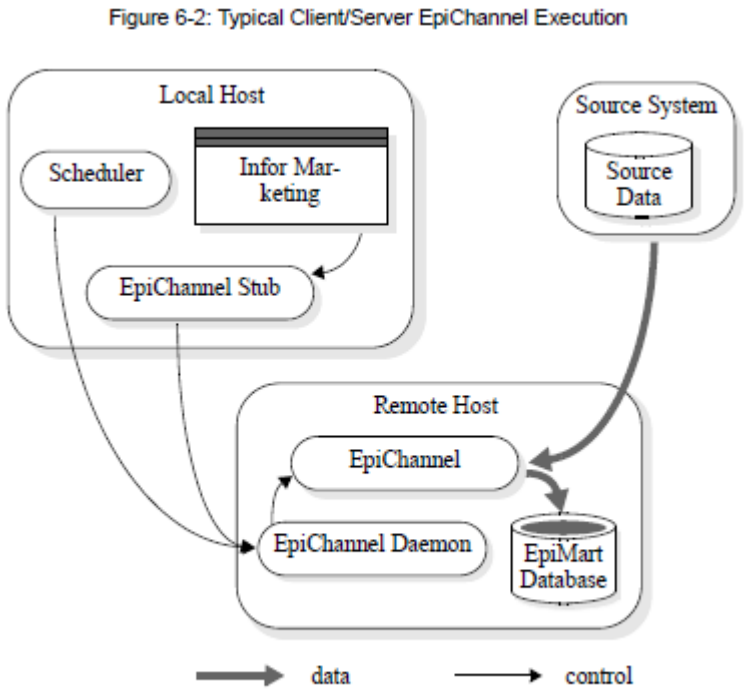


Figure 66: Typical Client/Server EpiChannel Execution

Fact Roll-Off

If your fact table is populated with transactional data and you do not wish to keep fact data that is older than a specified value in your data mart, you can choose to roll off fact data. You roll off fact data by setting roll-off options in the **Options** tab of the Fact dialog box (see "Fact Tables" on page 120) and running an extraction job that includes the special Roll-off extraction step.

If you are using an Oracle Enterprise Edition database server with the **Oracle Partitioning** option for your EpiMart database, fact roll-off is implemented using partitioned fact tables (see "Partitioning Oracle Base Dimensions" on page 36, "Partitioning an Oracle Fact Table" on page 41, and "Fact Tables" on

page 120) when possible. The roll-off extraction step rolls off all partitions that are older than the specified value.

If you are using any other database server for your EpiMart database, fact roll-off is implemented by deleting outdated rows. The roll-off extraction step deletes all fact-table rows that are older than the specified value.

Fact Roll-Off and State-Like Data

In most cases, roll-off must not be used for fact tables that are populated with state-like data. Since state-like data is recorded as an initial transaction followed by a series of difference transactions, fact roll-off can delete the original transaction while leaving difference transactions in the fact table. If a fact table contains only difference transactions for a state-like fact, without the initial transaction that records the starting state, queries related to these state-like facts can yield incorrect results.

In some cases, it is possible to use fact roll-off with fact tables that are populated by state-like semantics. If you are using state-like semantics to change historical data, rather than to record changing states (like inventories) over time, the safe roll-off of the historical data is possible.

Running a Fact Roll-Off Extraction Job

Before you run the Roll-off extraction step, you must ensure that, for every fact table, the mirrored A and B copies of the table contain identical data. You can do this by executing a job that truncates the fact staging tables, runs transactional semantics on the fact tables, and then runs the `AggBuild` and `MomBuild` extraction steps. When the A and B tables are identical, you can run a job that includes the Roll-off step to roll off fact data in the inactive mirrored table. After you have rolled off the inactive tables, you must swap mirrored tables and roll off fact data from the other set of tables. The default `Fact Rolloff` job (see "The Fact Roll-Off Job" on page 323) contains the correct sequence of steps to perform all of these actions. Infor recommends that you use the `FactRolloff` job to perform fact roll-off.

Data Stores

Data flow in an extraction job is configured using data stores. A data store is a logical location of data that functions either as a source or as a sink within an EpiCenter. Data stores include relational databases, as well as files and directories.

The concept of a data store is integral to the Infor Campaign Management extraction process. A data store represents the physical location of a data source from which data can be read or to which data can be written by Infor Campaign Management. Typically, the input data store is a source-system database or file, and the output data store is the EpiMart database, which holds data mart data. As

part of the Infor Campaign Management installation procedure, you create new, empty databases for the EpiMart, EpiMeta and EpiOp databases.

For more information about data stores, see "Data Stores" on page 203

Extraction Jobs

The Infor Campaign Management extraction process works through the running of jobs . A site defines the jobs needed to extract its data and runs these jobs on a regular basis to update the EpiMart database. A job is an ordered set of SQL commands, semantics, system calls, or other actions that must be performed as a single unit. Different kinds of job steps (such as system calls or SQL extraction commands) can be interspersed in any order. (See "Figure 67: Job Workflow" on page 179 and "Figure 68: Structure of a Typical Job" on page 180.) Each job step is executed against a particular source and destination data store.

Defining and Running Jobs

Admin Manager provides a graphical user interface (GUI) for defining job steps, along with input and output data stores. The actual job steps, including extraction commands and system calls, need to be customized for each site.

After the jobs are defined and the EpiCenter schema generated, the database administrator can invoke the EpiChannel command to begin the extraction process that populates a new EpiMart database or update an existing one. The EpiChannel command is normally invoked by scheduling or running a job in Admin Manager.

Job Steps and Extraction Steps

As shown in "Figure 67: Job Workflow" on page 179, an extraction job is a sequence of job steps . The actual work of extraction is done by extraction steps , which exist independently of any particular job. Job steps specify the extraction steps that are to be executed in a job and the order in which they are to be executed.

Extraction steps are defined as global objects for an EpiCenter. After you define an extraction step, you can reuse it in other jobs. Job steps are local to a specific job. You define extraction steps in the Extraction Steps dialog box, and you define job steps in the **Job Steps** tab of the Job dialog box.

Figure 6-3: Job Workflow

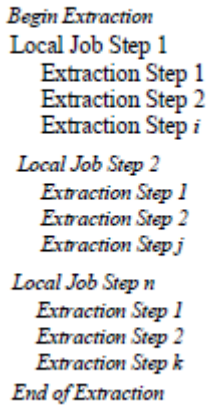


Figure 67: Job Workflow

Extraction Steps

An extraction step can be any one of the following:

- An extraction command

  Extraction commands are used to copy data from source databases into a staging table or an external table.

- A global extraction group

  Extraction groups , which are also called global groups , are ordered collections of extraction steps.

- A semantic instance

  Semantic instances are used to merge data from staging tables into fact or dimension tables.

- A system call

  System calls are used to execute command-line statements.

-

- An extraction utility

  Extraction utilities perform such tasks as aggregate building, campaign-management optimization, truncation and fact roll-off.

  Note: An extraction step exists independently of any job.

Job Steps

A job step is an extraction step that is included in a job, either as a local job step or as part of a global extraction group that is included in the job as a local job step. As shown in the job work flow (see "Figure 68: Structure of a Typical Job" on page 180), a local job step can be any one of the following:

- An extraction step

  Extraction steps that are included in a job are considered to be local job steps in that job.

  If you include a global extraction group in a job, then the global group is a local job step in that job. The extraction steps in this global group are executed as part of the job, but they are not local job steps in the job.

- A local group

  Local groups are ordered collections of local job steps that are local to a single extraction job.

Each local job step has associated input, output, and logging data stores. If you do not assign data stores to a local job step that is a member of a local group, then that job step uses the data stores that are assigned to the group. You must always assign data stores to the root job step. Members of global extraction groups always use the data stores that are assigned to the group.

Each job has a root-level local group that has the same name as the job. Additional local groups can be added under the root local group, or under any other local group that you have already added.

Figure 6-4: Structure of a Typical Job

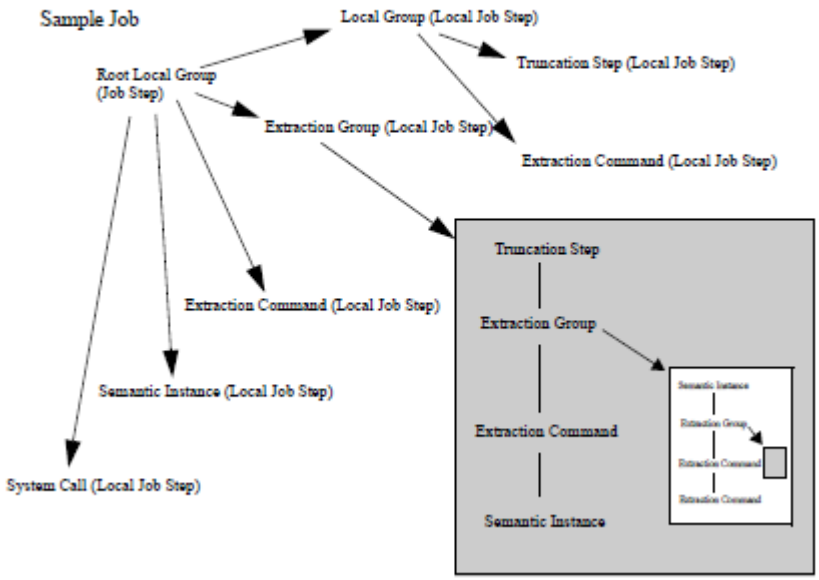


Figure 68: Structure of a Typical Job

Within a job, different local job steps can use of same extraction step, applying the same logic to different data stores, if the data stores associated with the local job steps have common structures. For example, the same extraction step can be used to extract data from an active transactional database and from an archival database of past transactions, as long as the archival data is in the same format as the data in the active database.

Note: Do not use an extraction step that uses the memorized values with more than one data store. (See "How EpiChannel Identifies Data To Be Extracted" on page 362.)

Restartability of Extraction Jobs

A lengthy extraction job can take several hours to complete. If a job fails after completing a significant amount of extraction, it is desirable to have the job repeat as little of that work as possible when the job restarts. You can configure an extraction job to restart automatically if extraction fails or to be manually restartable.

When you configure an extraction step, you specify the action to be taken if that step fails. An extraction job can be configured to restart at the failed extraction step, at the beginning of an extraction group that contains the failed extraction step, or at the beginning of the job.

For example, if a Semantic step fails, you need to have the job restart at the failed step. Since semantics creates no changes to staging tables, the extracted data in the staging tables is not compromised if a semantic extraction step fails. The semantic can have merged some rows of the staging table into the fact or dimension table that it is populating, but since most semantics ignore duplicate rows, it is usually safe to re-merge the entire staging table.

On the other hand, if an extraction command fails, the target staging table can be in an indeterminate state. If the extraction are restarted at the failed step, then the newly extracted data can be appended to the data that had already been extracted when the extraction command failed. To avoid this situation, you can create an extraction group that contains a truncation step for the staging table and the extraction command for the table. You can then configure the job to restart at the parent of the extraction command and truncation steps (that is, at the group that contains the two steps).

Thus, if the extraction command fails, then the group that contains it is repeated. When this group is repeated, the staging tables are truncated again by the truncation step and then all data is re-extracted by the extraction command. In this way, the integrity of extracted data is preserved, but previous extraction steps that had completed successfully do not need to be repeated. If an extraction job fails during aggregate building, any aggregates that are successfully built before extraction failed do not need to be rebuilt. When the AggBuilder program is restarted, it automatically builds only those aggregates that have not yet been built.

Extraction Commands

The two kinds of extraction steps that are used to move data from source databases into an EpiCenter are extraction commands and semantic instances. An extraction command extraction step can be either a stand-alone SQL statement or a pull/push extraction command.

- A stand-alone SQL statement is executed against either the source or the destination data store and is usually specific to the data store's type. These SQL statements are usually issued to achieve a side effect, such as dumping the transaction log, setting up environment variables, or executing a DDL statement. Any returned results are discarded.
- A pull/push extraction command is a type of extraction step that issues SQL against an input data store and is interpreted according to the rules of that database's engine. The returned (or pulled) results are inserted (or pushed) into the destination table for that step, which is either a staging table or an external table in the output data store of the job step.

- Pull/push extraction commands can be defined with the graphical extraction-command builder or with a SQL statement (see "Extraction Commands" on page 272). If an extraction command is defined with an SQL statement, this SQL statement must be written using the Infor Campaign Management' vendor-independent macros (see "Infor Campaign Management Macros" on page 441) if possible.

The result set columns are mapped either to columns in the destination tables or to other functions. The mapping is a case-insensitive exact match by name of the available columns, or function results, to the destination table columns.

At this stage of the extraction, each destination-table column must have a matched value. If not, the mapping fails, the extraction-command results are discarded, and the command fails. Extra columns that are available from an SQL statement or functions used to define the extraction command, but which are not used in the destination system, are not considered to be errors.

After they are mapped, the rows from the extraction command are fetched into EpiChannel memory, where they are forwarded to any function that consumes the field. The mapped results are forwarded to the destination database.

All operations on the destination data store or the EpiMart data store occur through the appropriate native library (API) for that data store. The inserts are batched so that the extraction statements maintain a count of rows fetched, rows forwarded, and rows committed. The batch is fully sent and committed before the extraction command ends and the next job step can begin.

Any error, as determined by the database engine evaluating the SQL, results in the extraction command being considered in error. If the On Error condition for the extraction command (as selected in the **Advanced** tab of the extraction command dialog box) is to abort, then the job halts without executing any other SQL operations or job steps.

SQL Macros

An extraction SQL statement can contain macros that are expanded before the SQL is executed. Some of these macros provide database-independent functionality by expanding in different ways for different databases. Other macros are expanded based on values recorded in your metadata, allowing your SQL extraction code to automatically reflect changes that you have created with Admin Manager.

Infor provides an extensive set of macros, which are described in "Appendix AA, "." You can also define custom macros, as discussed in "Defining a Macro" on page 446.

The UNKNOWN Dimension Row

When a fact staging table is loaded during an extraction, its dimension source system key columns (columns names that ends with `_sskey`) normally refer to rows in the base dimension staging tables. These source system keys are the actual values used to form the relationships on the source system between the fact and dimension source tables. That is, the `dimension_name_sskey` column of a fact staging table row refers to a dimension staging table row for the `dimension_name` dimension that has the same value in the `dimension_name_sskey` column.

Some source systems do not enforce referential integrity, however, which results in source system fact tables that have “dangling references,” or references to non-existent dimension values. Another possible source of a dangling reference is an optional or null foreign key in the source system. In the EpiMart database, all fact dimension keys must point to valid base dimension entries, and there must be no dangling references.

To prevent dangling references, each base dimension in the EpiMart database is populated automatically with a single special row called the `UNKNOWN` row . This row is always available for mapping fact dimension keys that are otherwise not point to a valid base dimension row.

For mapped dimensions, EpiChannel automatically remaps dangling references in fact staging tables to the `UNKNOWN` row. This remapping is not performed for unmapped dimensions.

Note: The sskey values 1 and '1' in unmapped dimensions are reserved by Infor for the UNKNOWN row.

Referring to the UNKNOWN Row during Extraction

When writing fact-staging extraction commands, you can refer to this special row by populating the `sskey` dimension role column with a string that is not used as an `sskey` in the dimension. By convention, the string `UNKNOWN` is used for this purpose whenever `UNKNOWN` not appears as a valid `sskey` value. Since null values are not permitted in staging tables, all null values must be replaced with such a string. This replacement is completed with the `$$NVL` macro (see "Appendix A, “Infor Campaign Management Macros,”" on page 441 for a complete description of the platform-independent Infor Campaign Management extraction macros):

```
SELECT
. . .
$NVL [ cust_code ~,~ 'UNKNOWN' ] customer_sskey
. . .
FROM
. . .
```

Note: If the string `UNKNOWN` appears as a legitimate sskey value in a dimension table, you must use a different value to refer to the `UNKNOWN` row. You can use any string value that cannot appear as a legitimate `sskey` value to refer to the `UNKNOWN` row.

Specifying Values for the UNKNOWN Row

Like all other dimension rows, the `UNKNOWN` row has actual values in all columns. By default, every dimension column with a character type assumes the literal value `UNKNOWN` , and every dimension

column with an integer type assumes the value 0. However, you can use Admin Manager to override this default value so that end-user queries return a different value. To do so, enter the desired value in the Dimension Column dialog box (shown in "Defining Dimension Columns" on page 104). This value is then used instead of the default. Alternatively, use the **Data Types** tab of the Configuration dialog box to change the default for a physical type.

SQL Limits for Facts

When configuring your data mart, keep in mind the maximum and minimum values available for your chosen data types. "Appendix C, “Data Type Values,”" on page 503 lists the database data types that correspond to the Infor Campaign Management physical types. Please refer to your database vendor’s system documentation for the limits on these data types. The selection of a data type that is not large enough to store your data affects the accuracy of Infor Campaign Management extractions and queries.

If necessary, you can define additional data types in the **Data Types** tab of the Configuration dialog box. See "The Data Types Tab" on page 157 for information.

Staging Tables

The most common destination for an extraction command is a staging table. As mentioned on page "Data Loading" on page 171, a staging table contains all of the fields required by a single base dimension or fact table. Unlike the actual fact and base dimension tables in the EpiMart database, which often use Infor Campaign Management-generated keys, the staging tables always have dimension-key columns that contain key values from the source system. Staging tables typically contain data extracted by only one job, rather than accumulating data over time.

EpiChannel integer maps the data that populates a staging table

If you are using dimension fusion (see "Fusion and Fission" on page 263), you extract fusion data to the fusion staging tables, from which the fusion tables are populated.

If you are using dimension data deletion (see "Dimension Data Deletion" on page 234), you extract source system keys that are to be deleted to the dimension delete staging tables, from which the dimension delete tables are populated.

After a staging table is populated, a semantic instance is used to merge the new data in the staging table into the existing data in the data mart. The semantic instance maintains the integrity and additive nature of the columns in the fact tables and maintains correct references to dimensions, even if the values of dimension elements change from one reporting of a fact to another. Semantic instances are created by Infor Campaign Management based on the structure of the destination table and the selected semantic type.

See "Extraction Commands" on page 272, for more information on populating staging tables and "Semantics," on page 223 for more information on built-in semantic types.

Note: In some cases, you may be able to use Streaming Extraction, which bypasses the staging table and populates the fact or dimension table directly. See "Streaming Extraction" on page 172, for more information.

Pre-Staging Tables

EpiChannel performs integer mapping, fact lookup, and other data optimizations when it populates a staging table. These processes, however, preclude the use of a data-cleansing ETL (Extraction, Transformation, and Loading) tool on your data. Pre-staging tables allow you to use an ETL tool before EpiChannel loads the staging tables and performs data optimization.

To use a pre-staging table, use your ETL tool to load the PSTG table that is created for each dimension. When these tables are fully loaded, use another job step to load the regular staging tables with a `SELECT * FROM $$DIM[]PSTG` statement. All EpiChannel optimizations are performed during this second step.

External Tables

External tables are database tables that are available to the EpiCenter, but that are not part of the EpiCenter star schema. External tables can be tables that you create, generally in the EpiMart data store, or tables that you have imported from any available data store.

Imported external tables are used by the EpiChannel graphical extraction tool to identify source-system tables. When a source-system tables is imported as an external table, the EpiChannel graphical extraction tool can display the structure of that table without connecting to the source database.

Extraction commands are sometimes directed to load into external tables rather than staging tables. External tables can serve as intermediary tables when more complicated multi-staged extraction is required, such as for grouping data, dividing values in a field into bins, or joining with data from another source system. They can also serve as holding fields used by non-Infor Campaign Management data transformation tools, such as third-party name and address cleansing tools. In the case of cleansing tools, extraction commands are then used to merge the cleansed names and addresses with other tables extracted from the source databases into the EpiMart tables.

Every EpiCenter includes a built-in external table called `last_extract_date` that is used to record the date of the last extraction. This table, which is located in the EpiMart database, is discussed in "End of Extraction" on page 186.

See "Using External Tables as Inputs to Staging Queries" on page 292, for more information about extracting into external tables. See "Importing External Tables" on page 216, and "Using the Graphical Extraction-Command Builder" on page 277, for more information about using external tables as source tables.

Beginning of Extraction

By default, every EpiCenter includes an extraction group called **Begin Extraction** . This extraction group consists of the Cancel In-Progress Job extraction step, which cancels any other extraction jobs that have not run to completion.

End of Extraction

By default, every EpiCenter includes an extraction group called **End of Extraction** . This extraction group consists of the following steps:

- Execute an extraction step that determines the date of the last extraction to be displayed to users.
- Toggle the current EpiMart table values for all fact and dimension tables that are changed.
- Create views that refer to the Current versions of all EpiMart tables.
- Direct the Infor Campaign Management Server to reload metadata from the EpiMeta database.

Step 1 on page 186 consists of an extraction command called `Max of MaxDate` that sets the last extract date based on the values in the external table named `last_extract_date` . You can add extraction commands that populate `last_extract_date` with appropriate values to the `MaxDate` extraction group, and you must ensure that extraction steps to populate `last_extract_date` (such as those in the **MaxDateMasterReplace** extraction group) are included in your job before the **End of Extraction** group. The **last_extract_date** table is included in your EpiCenter by default, and it is recommended for use in every installation.

You can use the **MaxDateMasterReplace** extraction group to repopulate the **last_extract_date** external table master row with the last extract date at the job execution date and time level. See "The Default Job" on page 319 for information. If you are writing custom extraction commands to populate the `last_extract_date` table, use the `$$TO_EPITIME` macro to convert datetime values to the appropriate format for use by the `Max of MaxDate` extraction command.

Note:

The last extraction date setting is the date and time that appears as the Results as of... date in reports. If you only wish to display the date of the last extraction, you can modify the `SQL: Max of MaxDate` extraction step to disable time display. To make this modification, replace the following line of code:

```
SELECT $$TO_EPITIME[MAX($$TO_TIME_FROM_EPITIME[max_date])]
```

Replace this line with the following line:

```
SELECT $$TO_EPIDATE[MAX( $$TO_DATE_FROM_EPIDATE[max_date])]
```

You must also ensure that the code that populates the `last_extract_date` table uses EPIDATE rather than EPITIME values. After you have entered these changes, only the last extract date is recorded the next time that the Max of MaxDate extraction step is run as part of an extraction job. The Infor Campaign Management Server then displays the last extract date to users.

Step 2 on page 186 of the **End of Extraction** group updates the current data mart table values in metadata, toggling between A and B for all facts and dimensions that are changed. Switching the focus between the mirrored tables involves a simple update to the metadata. No tables or databases are actually moved. See "Data Mart Mirroring: A and B Tables" on page 191 for a discussion of mirrored data mart tables.

Step 3 on page 186 creates database views that can be used for external reporting (see "Views for External Reporting" on page 197).

Step 4 on page 186 of the **End of Extraction** group directs the Infor Campaign Management Server to refresh cached metadata information from the EpiMeta database. Since the Infor Campaign Management Server uses this cached metadata information to determine which mirrored tables to use for queries, this step causes the Infor Campaign Management Server to direct all future queries against the tables that contain newly extracted data.

Semantic Instance Extraction Steps

Semantic instance extraction steps implement the data-merging phase of extraction, in which data is moved from staging tables into the EpiMart's permanent base tables. The process of moving data from staging tables into the EpiMart's base tables is called semantic transformation . Proper configuration of the data merging phase of extraction requires an understanding of the underlying meaning, or semantics, of the data.

It is also possible to load data directly into a base dimension or fact table (see "Streaming Extraction" on page 172). When you bypass the staging table in this way, you must apply a streaming semantic instance, which acts directly on the destination table.

For a full discussion of semantics, see "Chapter 8, "Semantics"" on page 223.

Extraction Utilities

Infor Campaign Management includes several special-purpose extraction steps that perform specific EpiCenter optimization and maintenance tasks. The following extraction utilities are available for use as extraction steps:

- AggBuild

  This extraction step builds aggregates that you have defined on your EpiCenter. This step is normally run as part of the **Builders** extraction group.

- Cancel

This extraction step cancels any other extraction jobs that are currently in progress with this EpiCenter. This step is normally run at the beginning of an extraction job as part of the **Begin Extraction** group.

- Cancel Backfeed

  This extraction step cancels any other backfeed jobs that are currently in progress with this EpiCenter. This step is normally run at the beginning of a campaign backfeed job as part of the **Begin Campaign Backfeed** group.

- Commit

  This extraction step commits an extraction job by making all Next tables Current. This step is normally run at the end of an extraction job as part of the **End of Extraction** group.

- Commit Backfeed

  This extraction step commits a backfeed job by making all Next backfeed tables Current. This step is normally run at the end of a backfeed job as part of the **End of Campaign Backfeed** group.

- Create Current Views

  This extraction step updates views on the EpiCenter. This step is normally run at the end of a job as part of the **End of Extraction** group or the **End of Campaign Backfeed** group.

- Dimension Backfeed Semantics

  This extraction step updates backfeed dimension tables for all backfeed-enabled base dimensions by merging new data from the backfeed exmadeport tables, which are populated whenever a campaign is run, into the main backfeed tables. This backfeed data can then be available to Infor Campaign Management users by extracting it into data mart tables.

- Fact Backfeed Semantics

  This extraction step updates backfeed fact tables for all backfeed-enabled base dimensions by merging new data from the backfeed export tables, which are populated whenever a campaign is run, into the main backfeed tables. This backfeed data can then be available to Infor Campaign Management users by extracting it into data mart tables.

- MomBuild

  This extraction step builds accelerators for lists and campaigns and indexes on your EpiCenter. This step is normally run as part of the **Builders** extraction group.

- Obtain Backfeed Lock

  This extraction step locks the backfeed export tables in preparation for execution of backfeed semantics. This backfeed lock prevents the campaign execution process from making changes to the backfeed export tables during backfeed semantic execution, preventing data inconsistencies.

- Refresh

  This extraction step refreshes the Infor Campaign Management Server.

- Release Backfeed Lock

  This extraction step releases the lock on the backfeed export tables after backfeed semantic execution, allowing the campaign execution process to once again make changes to the backfeed export tables.

- Rolloff

This extraction step rolls off historical fact data based on roll-off times that you have configured. See "..." on page 176.

- Rolloff Backfeed Semantics

  This extraction step rolls off historical backfeed fact data. Data that is older than the parameter specified in the `$$MAX_FACT_BF_ROW_AGE_IN_DAYS` macro is deleted from both A and B sides of the backfeed fact tables.

- Truncation

  This extraction step is used to delete all existing data from staging tables, external tables, or inactive base tables.

System Calls

SQL-based transformations are sufficient to extract from simple databases, sites with more complex databases requires multiple stages and additional commands. For example, you need to create lookup tables, gather data into ranges (binning), or detect duplicates. For some of these tasks, you can use system calls , which are executed during a job as if invoked from a command line on the platform on which EpiChannel is running.

In a multi-stage extraction, files might need to be moved, decompressed, decrypted, or purged. Although these steps can be placed before or after extraction statements, they need to be placed between SQL-based steps. EpiChannel coordinates the execution of system calls between extraction groups.

A complex job requires the following steps:

1. A system call to decompress a file.
2. The invocation of an extraction group that populates external tables.
3. A system call to invoke third-party software to cleanse the names in the external tables.
4. Execution of extraction groups that merge the cleansed names, along with data extracted from other tables in the original database and other databases, into the EpiMart database.

System Calls and Data Store Roles

Many system calls need to reference the location of databases or files. In the above examples, Steps 1 on page 189, 2 on page 189, and 3 on page 189 need to share the file and database names for the data they pass to each other. Although you can hard code these names, doing so is not recommended for these reasons:

- The system call can fail if the database login information changes and the system call text is not modified accordingly.
- The system call cannot be easily reused.
- Some file names need to be adjusted from run to run so that useful intermediate data is not overwritten.

Data-store roles (see "Data Store Roles" on page 205) provide an alternative to hard-coding database and file information in system calls. Data-store roles are referenced by macros in systems calls, and Infor Campaign Management use job configuration data to associate these data-store roles with actual data stores. When a system call is executed, all such macros are automatically replaced with references to the appropriate data stores.

For example, the system call

```
mkdir $$DIRNAME[WorkingDir]/myprogram_temp
```

can have the `$$DIRNAME` macro expanded to the MS-DOS or UNIX file path that EpiChannel is using as a working directory for the system-call job step. (WorkingDir is the data store role.) The `mkdir` command then creates a subdirectory in this location. Because EpiChannel generates a unique directory name for itself for each run, the above command creates a corresponding unique subdirectory for use by a program that you are executing, and this directory can be moved or purged along with the EpiChannel logs in the working directory.

A system call is executed from the working directory as if you opened a console window and changed directories to that location. Either use an absolute path to specify the command, or ensure that the value of the *PATH* environment variable includes the directory that contains the command executable.

`$$EPIBIN` expands to the path of the directory in which epichannel is located. If you are writing a system call that invokes a program called `mycommand.exe` and you place `mycommand.exe` in the same Windows directory as epichannel, the following command line can be used to invoke your program:

```
"$$EPIBIN/mycommand.exe" arg1 arg2 ...
```

If `mycommand.exe` is in a different directory, you must specify the full path to the directory when writing the system call.

Note: Use quotes as shown for "$$EPIBIN/mycommand.exe" arg1 arg2... DOS interprets the file name line, which expands to `C:\Program Files\Infor\Infor Campaign Management` in a default installation, as multiple command.com arguments unless the file name line is placed within quotes.

See "Appendix A, “Infor Campaign Management Macros,”" on page 441 for more information about Infor Campaign Management macros.

Note: All system calls are expected to have an exit code of zero, indicating success. You can use the Syscall dialog box in Admin Manager to set the On Error type for a system call to an appropriate value to indicate the action to be taken when the exit code is not zero.

Data Mart Mirroring: A and B Tables

Infor Campaign Management uses the metadata definitions of facts and dimensions to construct the physical EpiMart tables. Table-naming conventions indicate how the table is used. For instance, if a base fact table named `Sales` has been defined via Admin Manager, the system constructs the following tables:

Admin Manager

- `Sales_0_A`
- `Sales_0_B`
- `SalesStage`
- `SalesPStage`

`SalesStage` and `SalesPStage` are the staging table and pre-staging table that are used during extraction. The `_0` suffix on the other tables indicates that these are copies of the base fact tables, rather than aggregate tables.

For every fact and dimension table, the data mart contains two physical tables, with suffixes `_A` and `_B`. These tables contain “mirrored” copies of the dimension table.

The purpose of this table mirroring is to allow extractions to occur without disrupting the live usage of a system. At any time, either the A or B version of any table is current. Information about the current versions of all tables is stored in metadata, and end-user queries are run against the current versions of all tables.

If the current table for a fact or dimension is set to A, then end-user queries on that fact or dimension are run against the table that ends in A. The B table is inactive, and it contains data that is one extraction older than the data in the A table (that is, the B table contains the data that is current before the most recent extraction that affected this fact or dimension table).

Mirroring in Backfeed Tables

Every EpiCenter has two fact backfeed tables (`Communication` and `Message`) for each list-producing dimension, and three dimension backfeed tables (`Campaign`, `Cell`, and `Message`). These tables mirror the structure of the corresponding built-in fact and dimension tables.

Like the main data mart tables, the backfeed tables also have mirrored A and B copies. To avoid having to swap the A and B backfeed tables whenever a campaign is run, the backfeed tables include a set of tables called the P and Q backfeed export tables. These tables work in conjunction with the A and B backfeed tables, and either the P or Q table is current at any time.

Whenever a campaign is run, each of the current backfeed export tables (P or Q) is copied to the corresponding inactive backfeed export table, campaign data is added to the inactive backfeed export tables, and then the inactive tables are current. Whenever the A and B backfeed tables are toggled (usually at the end of the `Backfeed` job), the P and Q tables are also toggled. Therefore, backfeed data must be merged from these export tables into the main A or B backfeed tables whenever you toggle the main backfeed tables. This merging is normally completed by running the backfeed semantics

in the **Backfeed** job. The backfeed semantics are included in the default **Campaign Backfeed Dimension Semantics** and List **Producing Fact Backfeed Semantics** global extraction groups, which are included in the `Backfeed` job.

For best performance, and to prevent the amount of data from growing beyond the capacity of the disk volumes on which your EpiOp database resides, Infor recommends that you use the built-in **Backfeed Roll Off** extraction job to delete unneeded backfeed data, and that you schedule this job to run on a periodic basis. See "The Backfeed Roll-Off Job" on page 324, for more information.

You can also remove entries from the backfeed tables using the campaign undo feature that is available in the Infor Campaign Management front end. See the Infor Campaign Management end-user on-line help for a description of the campaign undo feature.

Swapping Mirrored Tables

During an extraction, the data in the current tables is never changed. For example, if the A tables are current for all fact and dimension tables, then the B tables are constructed by combining the A table data with any newly extracted data from the staging tables. In this way, end users can continue to query against the A tables, without knowing that extraction and aggregation are occurring simultaneously against the B tables.

When extraction finishes successfully, then the value of the current tables that is stored in metadata is toggled for all fact and dimension tables are changed. If there are no changes to a fact or dimension table, then the values for that table are not changed. In the above example, the current tables are set to B for fact and dimension tables that are changed. For fact and dimension tables that are not changed, the A tables continue to be current.

As soon as the Infor Campaign Management Server (described in chapter 10, “Infor Campaign Management Server,” in the Topic Implementation Guide) is notified that the data extract is successful, all end-user queries that refer to the changed facts and dimensions in this example are directed against the B tables. The A tables become inactive. The next time EpiChannel changes these facts and dimensions, the A tables are rebuilt and the letter A once again becomes current.

Updating Mirrored Tables

Each data mart table (that is, each A, B, P, or Q table) can be in one of four states: Current, Next, Previous, or Dormant. For each pair of mirrored tables (A/B or P/Q, for any fact, dimension, or backfeed table), exactly one table is Current at any time. The other table in the pair can be in the Next state, in the Previous state, or Dormant.

At the start of the first semantic instance in a job, all data mart tables that are not in the Current state are Dormant.

When a semantic instance populates a data mart table, it populates the Dormant copy of the table. When this table is successfully populated, the semantic instance places the Dormant table into the Next state. At the end of extraction, before the mirrored tables are swapped, all newly extracted data resides in tables that are in the Next state. Each pair of mirrored tables that are changed by the semantic

instance has one table in the Current state and one table in the Next state. Ordinarily, pairs of mirrored tables that remain unchanged have one table in the Current state and one table Dormant.

"Figure 69: Toggling Data Mart State in a Fact Table" on page 194 illustrates the data mart-state toggling process for a fact or dimension table.

The Commit Job Step

Extraction completes with the **End of Extraction** group. This group includes the Commit Job step, which performs the following actions for every fact, dimension, and backfeed table in a single atomic operation:

- If one of the mirrored copies of the table is in the Next state, then the copy of the table that is in the Current state is put into the Previous state.
- If one of the mirrored copies of the table is in the Next state, then that copy of the table is put into the Current state.

At the end of this operation, for every pair of mirrored tables that are changed, the changed copy of the table is in the Current state and the other copy (which is Current before the operation) is in the Previous state. For every pair of mirrored tables that are not changed, ordinarily the Current table remains Current and the other table is Dormant.

Rolling Back a Job

You can use Admin Manager to roll back a job that are committed. When you roll back a job, the following actions are performed for every fact, dimension, and backfeed table in a single atomic operation:

- If one of the mirrored copies of the table is in the Previous state, then the copy of the table that is in the Current state is put into the Next state.
- If one of the mirrored copies of the table is in the Previous state, then that copy of the table is put into the Current state.

After this operation, the data mart tables that are in the Current state before the job is committed are again in the Current state. The tables that are changed by the job are again in the Next state.

You can use Admin Manager to cancel a job that you have rolled back. When you cancel a job, all tables that are in the Next state are Dormant. When you cancel an extraction job, any data that is extracted in the job can no longer be available for querying.

You can use Admin Manager to commit a job that you have rolled back but that you have not cancelled. After you cancel a job, you cannot commit that job again.

The A and B tables that are in the Current, Next, and Previous states are listed in the **EpiMart State** tab of the Configuration dialog box (see "The EpiMart State Tab" on page 152).

Note: If the dimension backfeed tables are toggled by a job, then all dimension backfeed tables are toggled at the same time. Therefore, at any time, the current dimension backfeed tables are either all A tables or all B tables. Similarly, the current dimension backfeed export tables are either all P tables or all Q tables. Fact backfeed table states, however, can vary from table to table.

Note: All aggregates built in the tables are current after the job is committed. That is, if there is a table in the Next state, then aggregates are built in that table. If there is no table in the Next state, then aggregates are built in the Current table. See "AggBuild" on page 298, for more details.

Figure 6-5: Toggling Data Mart State in a Fact Table

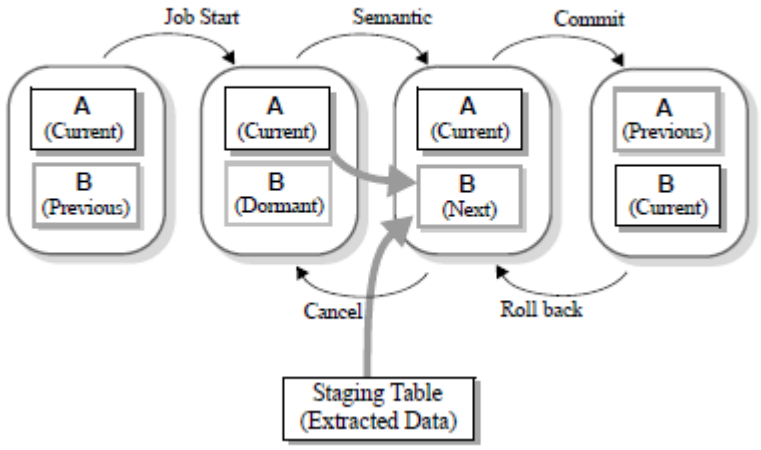


Figure 69: Toggling Data Mart State in a Fact Table

Incrementally Updating Fact, Dimension, and Backfeed Tables

In the case of fact, dimension, and backfeed tables with few updates, the existing tables are incrementally updated with new data. If many changes are needed for a table, then the Next tables are built by merging the Current tables with newly extracted data. This approach gives the best performance for large updates, since it avoids the cost of updating indexes and it can be performed as a nonlogged database operation.

Infor Campaign Management allows you to specify a change percentage below which fact and dimension tables are updated rather than being rebuilt. In order to enable incremental updating of tables, EpiChannel creates difference tables during extraction. These difference tables contain new and changed data from the staging table.

A table can only be incrementally updated if the EpiMart database contains difference tables from both the current and the previous extraction. In addition, the inactive mirrored table must contain data that is current before the previous extraction. All updates are used to the inactive mirrored table, in order to preserve the integrity of the data in the Current table.

The semantics determine the total number of rows in the difference tables from the current and previous extractions and compare that total to the number of rows in the current table. If the number of rows in the difference tables is below a specified percentage of the number of rows in the current tables, then the semantics attempt to perform an incremental update. The percentage below which a table can be incrementally updated is specified by the `$$DEFAULT_REVISION_PERCENTAGE` macro.

In an incremental update, the inactive mirrored table is updated with the data from the difference tables from the previous extraction. This creates a copy of the Current table. This table is then further updated

with the difference tables from the current extraction, yielding an updated table. The incremental update process is illustrated in "Figure 70: Incremental Updating of Fact and Dimension Tables" on page 195.

Incremental Partition Updates

If you are using a partitioned fact table on an Oracle database server with the Oracle Partitioning option (see "Partitioning Oracle Base Dimensions" on page 36 and "Partitioning an Oracle Fact Table" on page 41), then a single partitioned difference table is created. Each partition is revised or rewritten based on the number of changes in the data in that partition.

Note: Unlike fact partitions, dimension partitions are not tracked independently for mirrored state information.

Figure 6-6: Incremental Updating of Fact and Dimension Tables

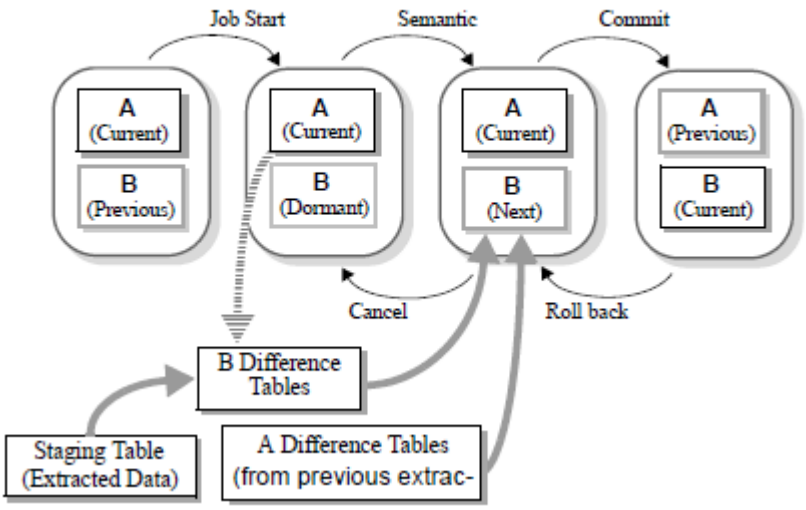


Figure 70: Incremental Updating of Fact and Dimension Tables

Incremental Update Semantics

In order for a table to be updated incrementally, you must use a semantic type that allows incremental updating.

Any dimension semantic type that has the **New Rows** option set to a value of **Any Update**, **New SSKey** , or **Primary Update** and that has the **Rewrite Rows** option set to **No Rewrite** allows incremental updating. The following built-in dimension semantic types allow incremental updating:

- First Dimension Value
- Latest Dimension Value
- Latest Dimension Value with Fusion/Fission
- Latest Dimension Value, Ignore Unknowns
- Latest Dimension Value, Preserve Fusion

- Slowly Changing Dimension
- Slowly Changing Dimensions, Ignore Unknown
- Slowly Changing/First Dim Value, Ignore Unknown
- Slowly Changing Primary, Latest Secondary

Any fact semantic type that has the **Row Type** option set to a value of **Transactional** , **Statelike** , **Transactional/Statelike** , or **Pipelined** and that has the **Use Current** option set to **Yes** allows incremental updating. The following built-in fact semantic types allow incremental updating:

- Pipelined
- Pipelined/Forceclose
- Statelike
- Statelike/Error on Stage Duplicates
- Statelike/SingleDelta
- Transactional
- Transactional/DeDup Stage
- Transactional/Statelike
- Transactional/Statelike/Force Close
- Transactional Incremental Streaming

The First/Last Fact and Reload Date Fact semantic types do not allow incremental updating.

The following additional conditions must be met in order for a table to be incrementally updated:

- The semantic being applied to the table is not marked as **Force Rebuild** in the Job dialog box.
- The previous extraction job did not apply an Initial Load semantic, and it successfully populated the difference tables.
- The most recent extraction job that applied a semantic to this table completed successfully and is not rolled back.
- The size of the difference tables is smaller than the specified percentage of the size of the Current table.

If these conditions are not met, then the table is rebuilt rather than revised.

Tablespaces for Mirrored Objects

If your EpiMart database resides on an Oracle or DB2 database server, you can configure different tablespaces for the mirrored A and B versions of objects, as described in "Physical Object Properties" on page 92.

On a DB2 database server, the use of different tablespaces for the A and B mirrored versions allows the use of the faster DB2 bulk load API during streaming extraction while still allowing end-user queries to be issued against the Current EpiMart tables.

Bulk Loading in DB2

On a DB2 database server, EpiChannel can extract data to EpiMart tables using either the InsertWriter or the LoadWriter method.

The InsertWriter method uses unlogged INSERT statements to write new data to EpiMart tables. This method is slower than the LoadWriter method.

The LoadWriter method uses the DB2 bulk load API to write new data to EpiMart tables. Infor recommends the use of the LoadWriter method, since it generally has significantly better performance than the InsertWriter method. In all supported verisons of DB2, when data is written to a table with the LoadWriter method, the tablespace containing the table is not locked, allowing it to remain available for end-user queries.

Considerations for Extraction on DB2

Additionally, in DB2 the tables created by the Query Engine for user queries are kept in memory after being logically dropped. The references to these logically dropped tables are kept in the `drop_table_tbl` table in both the EpiOp and the EpiMart. The tables are only physically dropped when the Infor Campaign Management Server is refreshed.

Views for External Reporting

If you wish to perform queries against EpiMart tables, you need to ensure that these queries are issued against the Current set of mirrored tables. In order to allow external queries to automatically reference the correct tables, the data mart maintains views of each Current fact table, each Current dimension table (with the exception of the built-in `Transtype` and `Date` dimension tables), and each Current backfeed table in the data mart.

The name of a basic fact or dimension view is derived from the name of a table by adding the `_0_V` suffix to the table name that appears in the data mart schema. For example, if Admin Manager lists a dimension table named `Customer` , the name of the external-reporting view for that table is `Customer_0_V` . This view always references the current table for the dimension. If any columns in the base dimension are integer mapped, this view shows the actual column values rather than the integer values generated by the data mart.

For every base dimensions, the data mart maintains an additional view of the raw data, with the suffix `_0_RV` . If the base dimension has no integer mapped columns, then this view is identical to the `_0_V` view. If the base dimension has integer mapped columns, then the `_0_RV` view shows the data mart-generated integer values rather than the actual column values. For a base dimension with integer mapped columns, queries on the `_0_V` view require joining the base table to the integer mapping tables for all integer mapped columns, so use of the `_0_RV` view can significantly improve query response time.

For every integer mapped base dimension column, the data mart maintains an additional view of the integer mapping that ends with the suffix `_CV` . For example, if your **Customer** table has an integer

mapped column named `favorite_cheese` , then the integer map view is named `Customer_favorite_25_CV`. Notice that the name of the column may be truncated due to limitations on identifier length, and a suffix may be added to uniquely identify the column. The `_CV` view has a column named after the original integer mapped column (`favorite_cheese` in the example) that contains the actual column value and a column named `KEY_INT` that contains the data mart-generated integer key associated with that column value.

All views and view columns are described in "Views" on page 527.

The Infor Campaign Management Scheduler

You can schedule jobs to be executed automatically with the Infor Campaign Management scheduler.

The Scheduler is responsible for running processes that are scheduled for automatic execution by Infor Campaign Management applications. Typical Scheduler tasks include running extraction jobs and generating campaigns.

The Scheduler divides tasks into queues , each of which can contain several different tasks. Within a queue, tasks are run based on their scheduled start time and a user-assigned priority level. Multiple tasks in a queue can be run in parallel. The Scheduler can run queues in parallel, and queues are scheduled based on queue schedule configurations and queue dependencies .

For more information about the Scheduler and queues, see "The Scheduler" on page 335

Campaign Output

When the Scheduler runs a campaign, it can export data to a database or generate output files that can then be used for campaign fulfillment.

Campaign Output Files

Scheduler output files are found in the export directory, which is specified by the `CampaignOutputDir` value in the **External Interfaces > Directory** registry key for the instance. By default, this directory is <Instance Root Directory>\Web\<instanceName>\AS\export.

For each campaign that is run by the Scheduler, one header file is written into the export directory. The name of this header file is datetime EXPORT.txt, where datetime is the date and time that the campaign runs. The header file contains the `campaign_sskey` associated with this campaign run, as well as the full paths of every export file.

The campaign output files are placed into subdirectories of the main export directory. These subdirectories are named after the output processors used for the campaign. Each output file is placed

in the directory named for the output processor that is associated with that file. The output files are named label datetime.extension , where label is the label specified by the end user, datetime is the date and time that the file is generated, and extension is the extension associated with the output file format.

Campaign files are generated as follows:

- The header file and all campaign output files are generated with an extension of .tmp.
- The .tmp extension is removed from all campaign output files.
- The header file is renamed from datetime EXPORT.txt.tmp to datetime EXPORT.txt.

If the Scheduler fails before completing campaign output, the export directory contains files with a .tmp extension. When the Scheduler restarts, these files are automatically renamed if the Scheduler determines that campaign export completed successfully before Scheduler failure, or deleted if the Scheduler determines that campaign export did not complete successfully. You can ensure that a campaign has run to completion by examining the extension of the header file for the campaign.

Note: If the header file for a campaign is named datetimeEXPORT.txt.tmp, then campaign export has not run to completion. You must not use any of the output files for that campaign, as the files can not be complete.

Campaign Output Tables

If a user chooses to export a campaign to a database table, Infor Campaign Management creates the tables in the EpiOp database. The header file in the campaign export directory contains the names of the export tables.

Campaign output tables have names of the form:

- EMXP_campaignID_listOrder_tableNumber

where

- campaignID is the ID of the campaign.
- listOrder is the output file number (starting with 1). You can have numbers higher than 1 if you have more than one output file defined for the campaign.
- tableNumber is a running counter of the tables belong to a particular output file.

For each output file, Infor Campaign Management generates one database table containing all of the demographic data and another table containing each transactional attribute. Table column names are based on the names of the measures and attributes to which they correspond.

The following is sample data from an export table:

			VMASHIP_DATE_YEAR

customer_key_real	VMBSUM_LINE_ITEM_QTY	VMARECEIPT_DATE_YEAR	
1151	204.00	1995	1995
650	107.00	1997	1997
131	44.00	1994	1994
477	8.00	1996	1996
473	50.00	1998	1997
495	143.00	1992	1992
491	28.00	1994	1993
668	607.00	1993	1993
833	354.00	1996	1996

1179	279.00	1998	1998

Campaign Undo

When a campaign is executed, records of that campaign are stored in backfeed tables (see "Backfeed Tables" on page 54). The existence of the backfeed tables allows users to analyze campaigns that they have run in the past, as well as exclude customers from receiving campaign treatments after having received other treatments in the recent past.

End users can choose to undo a campaign that is run in the past so that its results are not included in campaign analysis figures, and so that members of that campaign are no longer excluded from new campaigns.

When an end-user flags a campaign to be “undone,” an entry for that table is placed in the campaign_undo table in the EpiOp. The next time a Backfeed job is run, information about the campaign is removed from the Backfeed P, Q, A, and B tables. The campaign no longer exists in EpiOp tables.

If a Campaign extraction job runs after a campaign executes but before the campaign is undone, the campaign-related tables in the EpiMart also contain information about the undone campaign. Consequently, the Campaign extraction job includes steps that:

- Copy data from the campaign_undo table in EpiOp to a campaign_undo table in the EpiMart.
- Use information in the EpiMart’s campaign_undo table to debook undone campaign information from all campaign and message fact tables. Debooking this information effectively removes the campaign from the tables.

  Undone campaign information is not removed from the Campaign, Cell, and Message dimension tables in order to maintain referential integrity between base campaign tables and their aggregates. Consequently:

  - Count Distinct operations on campaign attributes include attribute values that are entirely debooked. You can eliminate this effect by using the **Eliminate anti-matter for count distinct** option (available from the **Advanced Options** area of a Rows and Columns report).
  - Min, Max, and Count operations on the Campaign, Cell, and Message dimensions include rows from undone campaigns.

Data Stores

7

A data store is a logical location of data to be used either as a source (input data store) or sink (output data store) within an EpiCenter. Typically, the input data store is a source system database or file, and the output data store is the EpiMart database on your RDBMS server.

Data Store Types

Each data store has a type that indicates the method of data transfer it uses. Recognized data-store types include:

- Microsoft SQL Server
- Oracle
- IBM DB2
- Generic ODBC data source
- File

As shown in "Figure 71: The Infor Campaign Management Back End" on page 204, the Infor Campaign Management application extracts raw data from a site's source system—for example, an RDBMS server, a generic ODBC source system, or a flat file—using Open Database Connectivity (ODBC) or a native database library. ODBC is an abstraction layer that provides a common interface to many databases. This data is read by various ODBC drivers and written using the target database system's API. Infor Campaign Management can access any data store for which an ODBC driver exists.

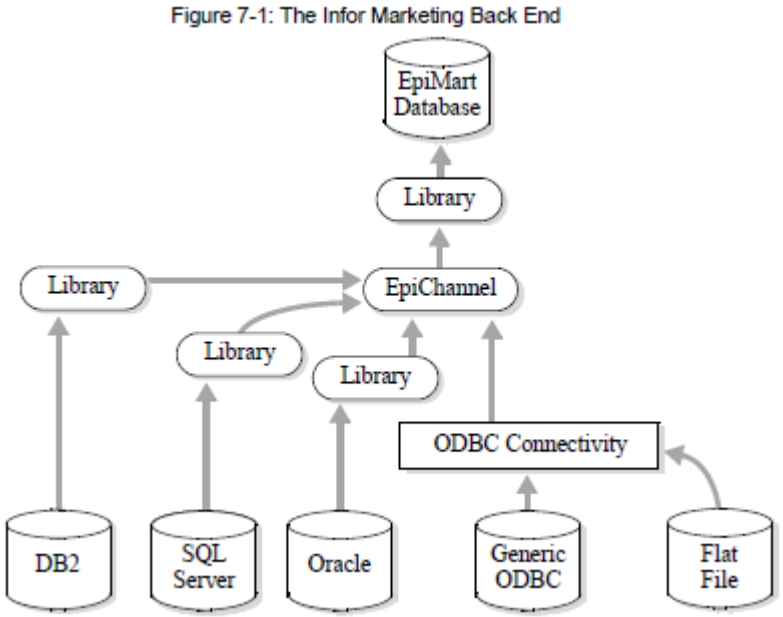


Figure 71: The Infor Campaign Management Back End

EpiChannel, the Infor Campaign Management extraction program that executes jobs and logs job activity, opens an ODBC or native connection to the source system and a database library connection to the target server and initializes tables.

Default Data Stores

You use Admin Manager to configure the data stores as part of setting up the extraction process at your site. When an EpiCenter is created, it has the following default data stores:

- **EpiMart** This data store references your EpiMart database.
- You can use the **Properties** tab of the Data Store dialog box to change the database that is used for your EpiMart database, as well as the username and password for that database. You can also change the **Connection Settings** in the **General** tab. You cannot change any other parameters for the EpiMart data store.

  Note: You must use the same user name/password combination as used during creation of the database during installation.

- **EpiMeta**: This data store references your current EpiMeta database. You can change the **Connection Settings** for this data store.
- **EpiOp**: This data store references your EpiOp database. The EpiOp database contains operational data for your EpiCenter.
- The EpiOp data store is used for runtime data, such as backfeed tables, query logs, saved lists, and EpiChannel logs. You can change the **Connection Settings** and **Properties** for this data store.

In addition to the pre-defined data stores, you need to define a data store for every source system from which you are extracting. You can set up as many data stores in Infor Campaign Management as necessary.

Data Store Roles

Data stores are associated with extraction jobs by means of data-store roles. The following data-store roles are provided by default.

- Input

  An input data store, typically a source system from which data is extracted.

- Output

  An output data store, typically your EpiMart database.

In addition, you can define data-store roles of your own choosing.

You use the Job dialog box to assign data stores to data store roles for a job. You can also assign data stores to data store roles for individual local job steps, overriding the global settings for the job.

For example, if a job extracts data from two source systems into your data mart, you can have a separate local job step for each source system. You can then assign different input data stores to these two local job steps.

You can also have a more complex job that extracts data from your source system into external tables in your EpiMart database and then moves data from the external tables into staging tables. In such a job, first you have to extract steps that use the source-system data stores as input data stores and the **EpiMart** data store as output data store. Since the external tables and the staging tables are both in the EpiMart database, the extraction steps that then move data from the external tables to the staging tables use the **EpiMart** data store as both input and output data store.

Data-store roles can be referenced by system-call macros in system-call extraction steps that you define. See "System Calls" on page 297, for a discussion of system-call extraction steps. See "System-Call Macros" on page 492, for a discussion of system-call macros.

Defining a Data Store

To define a new data store, right-click the **Data Stores** icon in the Extraction folder and select **New Data Store**. This action opens the Data Store dialog box. This dialog box has tabs named **General** , **Properties** , **Tables** , and **Usage** .

Figure 7-2: Data Store Dialog Box: General Tabs

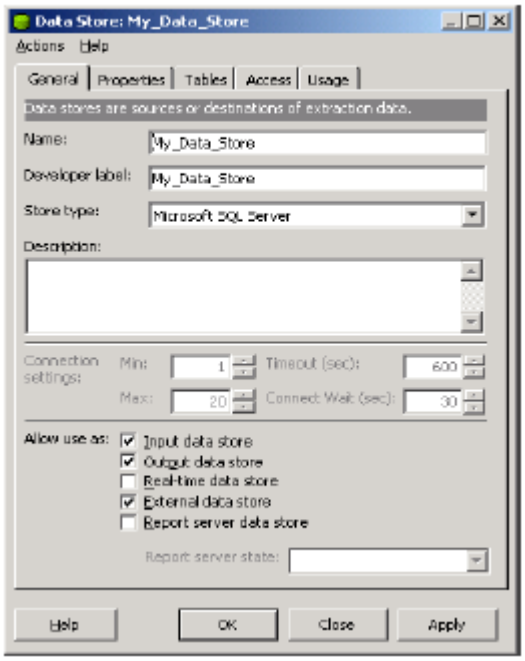


Figure 72: Data Store Dialog Box: General Tabs

Defining a Data Store

1. In the **General** tab of the Data Store dialog box, enter a name, developer label, and, if desired, a description.
2. Select the type of data store in the Store Type pane. The available options are: **Microsoft SQL Server**, **Oracle**, **Generic ODBC Data Source**, **File**, and **DB2**.

   If you choose the **File** data store type, you must have a flat-file ODBC driver configured. See the *Infor Campaign Management Installation Guide* for details.
3. Use the Allow Use As pane to specify the way the data store can be used:
   - **Input data store** allows the data store to be used as the input data store for a job step. If this option is unchecked, an error message is displayed if you attempt to use this data store for the Input data store role.
   - **Output data store** allows the data store to be used as the output of a job step. If this option is unchecked, an error message is displayed if you attempt to use this data store for the Output data store role.
   - **Real-Time personalization data store** indicates that the data store refers to an Infor Real-Time database (RTDB).

     Note: In order to enable CRM Interaction Advisor (Real-Time) reporting and deployment functionality in your EpiCenter, you must define a Real-Time data store.

If you are integrating Infor Campaign Management and Real-Time, the database code pages must the same on both the AS EpiMart database and the Real-Time database.

- **External data store** indicates that front-end users can use this data store for EDA (External Data Access) reports and campaigns. External data stores can also be configured from the front-end interface.
- **Report server data store** indicates the data store can be used to run reports.

4 The **Properties** tab allows the access specifications based on your data store type:

- For Microsoft SQL Server, enter the server name, database name, username, and password. With an SQL Server database server, you can select **Windows Authentication** to use Windows integrated security rather than SQL Server security.

  Note: If you use Windows integrated security for user authentication, choose Default (Same as EpiMeta) authentication for the EpiMart Data Store. Choose this same setting for all other data stores that use integrated security for user authentication.

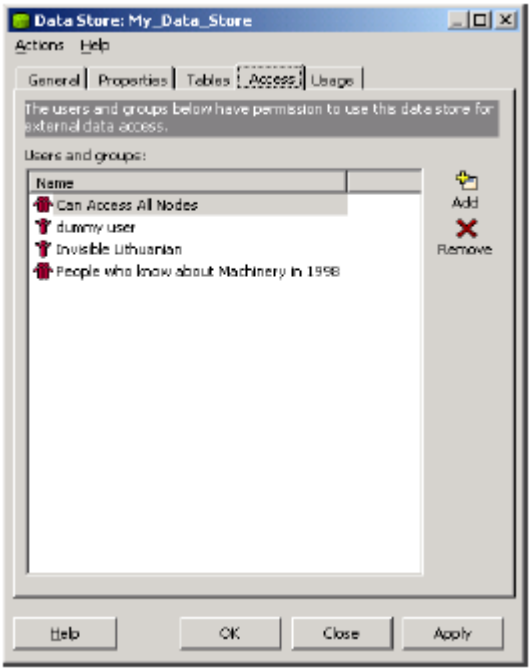
- For Oracle, enter the service name, username, password, SID, and port number.
- For DB2, enter the instance name, database name, username, and password, and the DB2 JDBC port number for your instance. Be sure to give each instance the same alias on all clients.

  Note: Infor Campaign Management uses the DB2 driver IBM Data Server Driver for JDBC and SQLJ. This driver supports both Type 2 and Type 4.

  Note: If you are using a DB2 or Oracle data store, ensure that you have set the appropriate code page environment variable. For DB2, the variable is DB2CODEPAGE. For Oracle, the variable is NLS_LANG. For more information, see the *Infor Campaign Management Installation Guide*.

- For a generic ODBC data store, enter the **DSN name** , select an **ODBC driver** from the drop-down list box, and optionally select a database type from the **DB Type** drop-down list box. For macros to expand properly you must also set up a DSN for the data store by using the Data Source Administrator in the Windows Control Panel or by adding an entry to your .odbc.ini file on a UNIX host.
- For a data store of type file enter the path for a directory or file.

5 If the data store has been configured as an external data store, you can give front-end users access to the data store on the **Access** tab. Click **Add** to select users and groups to whom you wish to grant access to this data store for EDA reports and campaigns. To remove access, select a user or group and click **Remove**.



6 Click **OK** to save your data store and close the dialog box.

If you wish to launch the Import Table wizard for this data store (see "External Tables" on page 213 and "Importing External Tables" on page 216), select **Import Tables** from the **Actions** menu in the dialog box. To launch the External Table Difference Report wizard for this data store (see "Creating a Difference Report" on page 217), select **Difference Report** from the **Actions** menu.

ODBC Data Stores

ODBC data stores are only supported as extraction source databases. They can be used in extraction commands that read in data from ODBC data sources and populate facts, dimensions, and external tables with this data. However, they are not supported for use as data stores for SQL execution job steps that do not populate these tables. Doing so results in a "no suitable driver" error. If you have job steps that execute SQL against a data store without populating fact, dimension, or external tables, configure these job steps to use SQL Server, Oracle, or DB2 data sources.

File Data Store Example

The following procedure is an example of how to set up a text file data store. This procedure and the one contained in "ODBC Data Store Example" on page 211both use the following sample text file called SampleIndivText.txt:

```
"Indiv_Id", "Indiv_Name","City ", "State ","Income"
```

```
"IID01 ","AMBER ","Belfast ", "DC ",10000
```

```
"IID02 ","BETTY ","Ottawa ", "OH ",20000
```

```
"IID03 ","CATHY ","Canyon Village ", "MS ",30000
```

```
"IID04 ","DAVIS ","Stirling ", "KY ",40000
```

```
"IID05 ","ELAINE ","Brookline ", "CO ",50000
```

```
"IID06 ","FRANK ","Waupaca ", "PA ",60000
```

```
"IID07 ","GEORGE ","Snake River ", "ME ",70000
```

```
"IID08 ","HELEN ","Fox Lake ", "NM ",80000
```

```
"IID09 ","IRINE ","Dallas ", "GA ",90000
```

```
"IID10 ","JASMINE ","Hana ", "ND ",99000
```

Note: All column data types are VARCHAR except for Income, which is INT.

Defining a File Data Store

1. Open the Extraction folder in your EpiCenter and click **New** to open the Data Stores dialog box.
2. In the **General** tab, do the following:
   a. Enter a name and developer label.
   b. Set the **Store type** to **File**.
   c. Set **Allow use as** to **Input data store**.
3. In the **Properties** tab, do the following:
   a. Browse to the folder that contains your source file.
   b. Do not enter a file name. (Leave this field blank.)
   c. Click **OK**. The newly created data store appears in the Data Stores folder.
4. Create an extraction job step for the source file. The following SQL example extracts data from sample file SampleIndivText.txt:

```
SELECT
Indiv_Id as Indiv_sskey,
Indiv_Name as Indiv_Name,
City as Indiv_City,
State as Indiv_State,
Income as Indiv_Income
FROM
SampleIndivText.txt
```

Note that macro's are not allowed in the SQL query.

ODBC Data Store Example

The following procedure is an example of how to set up an ODBC data store. See "File Data Store Example" on page 209 for the sample text file used in this example.

Note: Do not use an ODBC data source for SQL execution job steps. Instead, configure the job steps to use SQL Server, Oracle, or DB2 data sources instead.

ODBC data stores are only supported as extraction source databases. They can be used in the extraction commands that read in data from ODBC data sources and populate facts, dimensions, and external tables with this data. However, they are not supported for use as data stores for SQL execution job steps that do not populate these tables. Doing so results in a "no suitable driver" error.

Defining an ODBC Store

1. Open the Extraction folder in your EpiCenter and click **New** to open the Data Stores dialog box.
2. In the **General** tab, do the following:
   a. Enter a name and developer label.
   b. Set the **Store type** to **Generic ODBC data source**.
   c. Set **Allow use as** to **Input data store**.
3. In the **Properties** tab, do the following:
   a. Enter the name of the text file in the **DSN Name** field.
   b. Select Microsoft Text Driver (*.txt; *.csv) as the database driver.
   c. Set **Database Type** to **File**.
4. Click **OK** . The newly created data store appears in the Data Stores folder.
5. Create an extraction job step for the source file. The following SQL example extracts data from sample file SampleIndivText.txt:

```
SELECT
Indiv_Id as Indiv_sskey,
Indiv_Name as Indiv_Name,
City as Indiv_City,
State as Indiv_State,
Income as Indiv_Income
FROM
SampleIndvText.txt
```

Configuring the ODBC Connection in Windows

Use the following procedure to set up and configure the Windows ODBC connection necessary for using an EpiCenter ODBC data store.

1 Copy the source file to a folder on the local machine. The default ODBC folder is: `C:\Program Files\Common Files\ODBC\Data Sources`.

2 Navigate to **Start > Programs > Administrative Tools > Data Sources (ODBC)** to open the **ODBC Data Source Administrator** dialog box.

3 In the **System DSN** tab, do the following:

   a Click **Add** and select the same database driver as you did in Step 3 on page 211 of "Defining an ODBC Store" on page 211

   b Click **Finish** and enter the DSN name that you mention in Step 3 on page 211 of "Defining an ODBC Store" on page 211.

   c Uncheck **Use Current Directory** and browse to the ODBC folder that contains your source file.

   d Click **Options** , uncheck **Default** and select *.txt as the file extension.

   e Click **Define Formats** to open the Define Text Format dialog box.

   f Select your source file from the list of tables and check **Column Header Name**.

   g Select **Custom Delimited** from the list of format options and enter a comma (,) in the **Delimiter** field.

   h Click **Guess** to populate the column list.

4 Close all open dialog boxes to complete the ODBC configuration.

Connection Settings

The Infor Campaign Management Server creates a pool of connections for every data store to which it connects. For the EpiMart , EpiMeta , and EpiOp data stores, you can configure the following connection setting options in the **General** tab of the Data Store dialog box:

- **Min**: The minimum number of connections to maintain to the data store. For the EpiMeta data store, this must be set to at least 3 if the **Timeout** value is less than 30.
- **Max**: The maximum number of connections to open to the data store. This value must be at least 5.
- **Timeout (sec)**: The number of seconds of inactivity after which an open connection to a data store is closed. If the minimum number of connections is reached, no more connections are closed, even if the timeout value is reached. In most cases, the timeout must be set to a value of at least 30.

External Tables

External tables are tables that are not part of the data mart schema but that are available to the EpiCenter, generally for use in extraction. External tables can be located in any data store, including the EpiMart data store. All source data stores must be available when executing an extraction job, but you do not need to be connected to a data store when configuring an extraction command that extracts from external tables in that data store.

Extraction statements are sometimes directed to load external tables that serve as intermediary tables for multi-staged extraction. By default, an EpiCenter includes one external table called `last_extract_date`, which is located in the EpiMart data store. Use of this table is recommended, but optional.

See "External Tables" on page 185, for more information on external tables.

Defining External Tables

External tables are defined and edited in the data store dialog box. The **Tables** tab of the Data Store dialog box shows external tables that you have imported (see "Importing External Tables" on page 216) and external tables that you have defined. Imported tables are displayed with the **Imported** check box selected.

Defining an External Table

1 Open the Data Store dialog box for the data store in which you wish to define an external table and click the **Tables** tab.

Figure 7-3: Data Store Dialog Box: Tables Tab

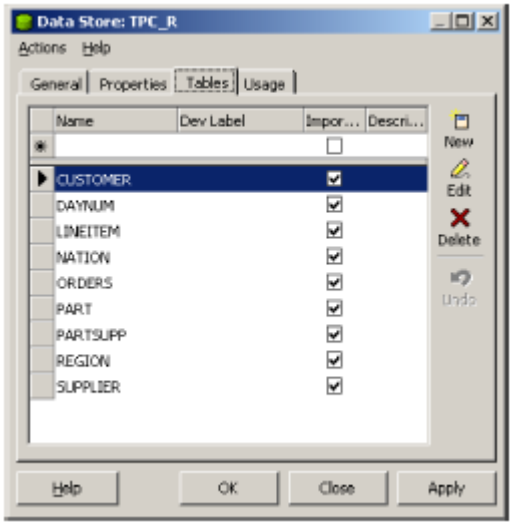


Figure 73: Data Store Dialog Box: Tables Tab

2 Enter the name and any description for the table in the grid control and press the Enter key to define the table.

The Infor Campaign Management schema generator can generate external tables in the EpiMart database. If you do not choose to have the schema generator create your tables, then you must use your database administration tools to create them. You must use your database administration tools to create external tables that you define in any data store other than the EpiMart data store.

Viewing or Editing External Table Definitions

You can view or edit the definition of an external table in the External Table dialog box. If a table is imported, you cannot add new columns or change the physical properties of existing columns in that table.

Editing an External Table Definition

1 Select the external table that you wish to edit in the **Tables** tab of the Data Store dialog box.

2 Click **Edit** to display the External Table dialog box.

Figure 7-4: External Table Dialog Box: General Tab

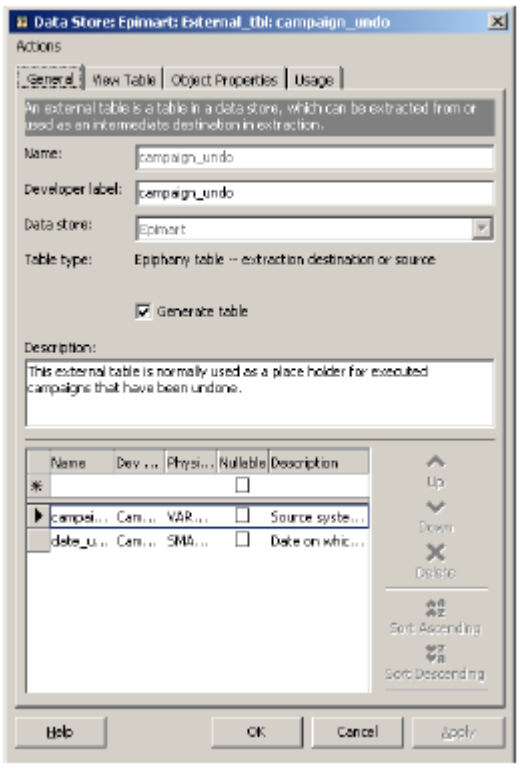


Figure 74: External Table Dialog Box: General Tab

3 In the **General** tab, check **Generate** if you wish to have the schema generator create the table. An external table must be in the EpiMart data store in order to be generated. If you choose to have the schema generator create the table, then you must specify the table columns.

4 To add a new column to the table definition, select the top row of the grid control in the lower pane of the External Table dialog box and do the following:

   a Enter the column name in the **Name** field.

   b Select the physical type of the column from the drop-down list box. See "Appendix C, “Data Type Values”" on page 503 for descriptions of these physical types.

   c If **Nullable** is checked, then the external column allows null values.

   d If desired, add a **Description** for your reference.

   e Press the Enter key or change focus to define the column. You can change focus by selecting any other field, clicking a button, or selecting a different tab of the dialog box.

You can also view and edit the actual data in an external table in the **View Table** tab of the External Table dialog box (see "Figure 75: External Table Dialog Box: View Table Tab" on page 215).

Figure 7-5: External Table Dialog Box: View Table Tab

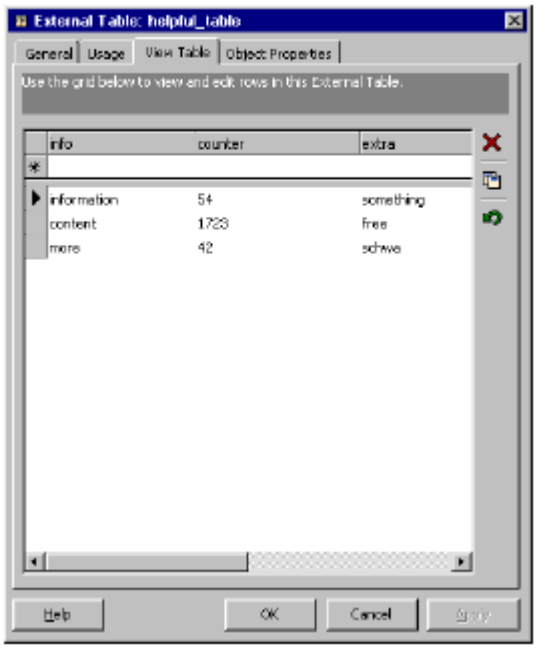


Figure 75: External Table Dialog Box: View Table Tab

Note: You cannot edit fields with a Date data type in the View Table tab of the External Table dialog box.

Note: If you have two or more rows with identical column values in an external table and use the View Table tab of the External Table dialog box to delete or update one of these rows, then all of the identical rows are deleted or updated.

Importing External Tables

In order to use the Infor Campaign Management graphical extraction tool, you must first import all required source-system tables as external tables. Note that this operation does not import the actual data in the source-system tables.

You use the Import External Table wizard to import external tables from a data store that you have defined. To start this Wizard, right-click the Extraction folder and select **Import External Table** , or select **Import External Table** from the **Extraction** submenu of the **Objects** menu.

On the first screen of the wizard, select the data store from which you like to import table definitions.

On the second screen (see "Figure 76: Selecting Tables in the Import External Tables Wizard" on page 216), select the tables or views that you wish to import. You can choose the types of objects that are available for import in the **Display from Source Database** drop-down list box. Select **Tables** to display only tables, select **Views** to display only views, and select **Tables & Views** to display both tables and views.

Figure 7-6: Selecting Tables in the Import External Tables Wizard

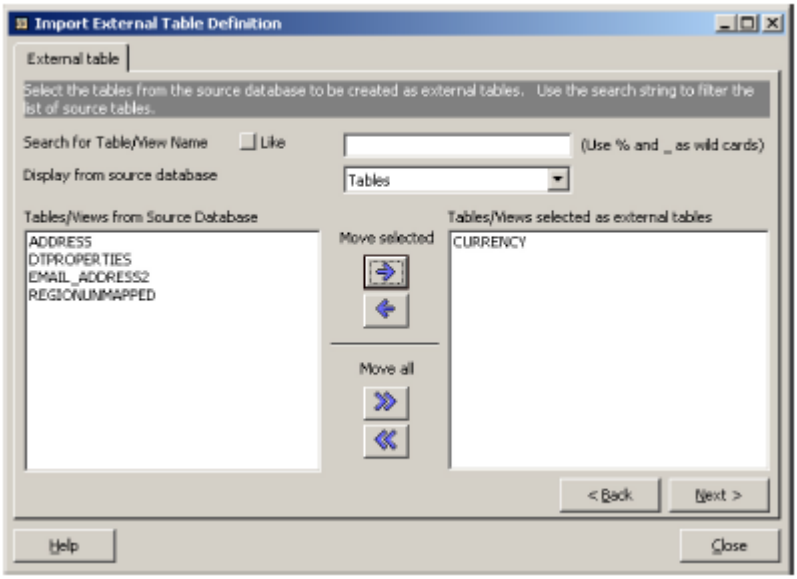


Figure 76: Selecting Tables in the Import External Tables Wizard

Use the arrow buttons to move selected tables and views from the Tables/Views From Source System pane to the Tables/Views Selected as External Tables pane. Use the double arrow buttons to move all displayed tables and views. You can use the search function to find names in the list of source-system tables and views.

Note: If the name of a source table can be changed in the future, Infor recommends that you create a view in your source database that refers to the desired table. You can then import this view as an external table. The administrator of the source database must ensure that the name of this view is not changed and that the view continues to refer to the desired table.

On the third screen (see "Figure 77: Editing Table Definitions in the Import External Tables Wizard" on page 217), you can edit the definitions of the external tables. Select the table for which you wish to edit

the definition from the **External Table Name** drop-down list box. Uncheck the **Use** check box for any table column that you do not need for extraction. Only columns with the **Use** check box checked are displayed in the graphical extraction-command editor.

Figure 7-7: Editing Table Definitions in the Import External Tables Wizard

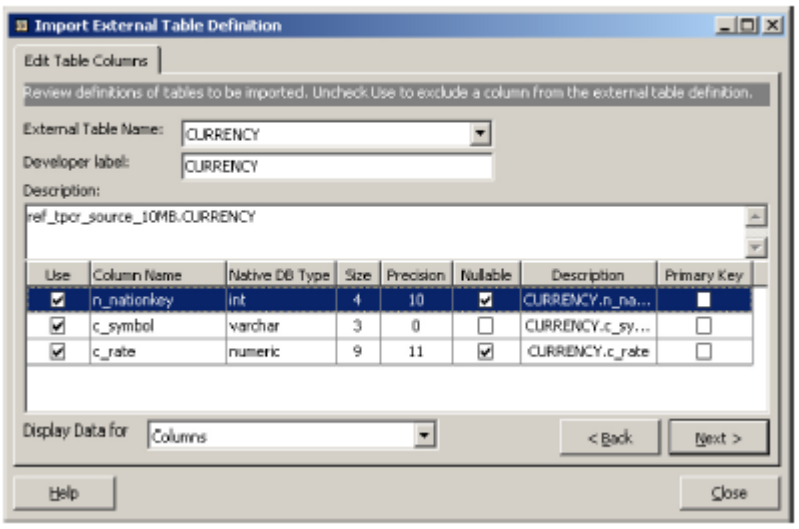


Figure 77: Editing Table Definitions in the Import External Tables Wizard

The fourth screen shows the definitions that is added to metadata. Click **Finish** to add these definitions.

Note: When you import an external table, Infor Campaign Management issues warning about source system foreign keys failing to be imported. These warnings occur because Infor Campaign Management attempts to import all primary and foreign keys so that table joins can be displayed in the graphical extraction command definition screens. If a foreign key references a table that is not imported with the current table or that does not already exist in EpiMeta, the foreign key export fails. This behavior is correct and the warning can be ignored.

Creating a Difference Report

External table definitions are stored in your EpiMeta database, allowing you to refer to these tables without connecting to the source database that contains the actual data. If the structure of the physical source tables is changed, you must ensure that the external table definitions are updated to reflect these changes.

An external table difference report allows you to compare external table definitions to the structure of the physical source tables, and to update your external table definitions to reflect any changes that have been made to the source tables.

Creating an External Table Difference Report

1 Right-click the Extraction folder of your EpiCenter and select **External Table Difference Report** to display the External Table Difference Report Wizard (see "Figure 78: The External Table Difference Report Wizard" on page 218.

Figure 7-8: The External Table Difference Report Wizard

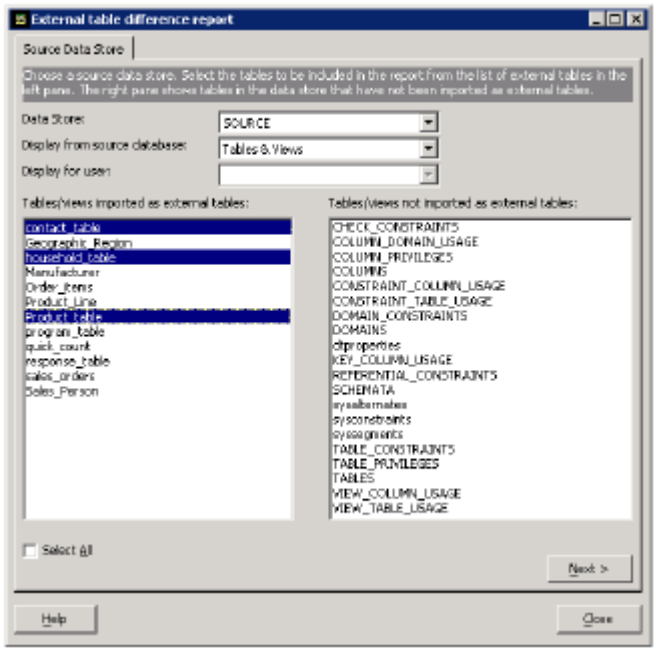


Figure 78: The External Table Difference Report Wizard

2 Select a **Data Store** from the drop-down list box. When you select a data store, external tables in that data store are displayed in the Tables/Views Imported as External Tables pane on the left and all other tables and views in the data store are displayed in the Tables/Views not Imported as External Tables pane.

3 In the **Display from Source Database** drop-down list box, select **Tables & Views** to display all tables and views in the data store. Select **Tables** to display only tables and **Views** to display only views.

4 In the Tables/Views Imported as External Tables pane, select the external tables for which you wish to create a difference report.

5 Click **Next** to generate the difference report. The difference report is displayed in the next pane of the wizard (see "Figure 79: An External Table Difference Report" on page 219

6 Select a table from the **Actual Table Name** drop-down list box to display the difference report for that table.

Figure 7-9: An External Table Difference Report

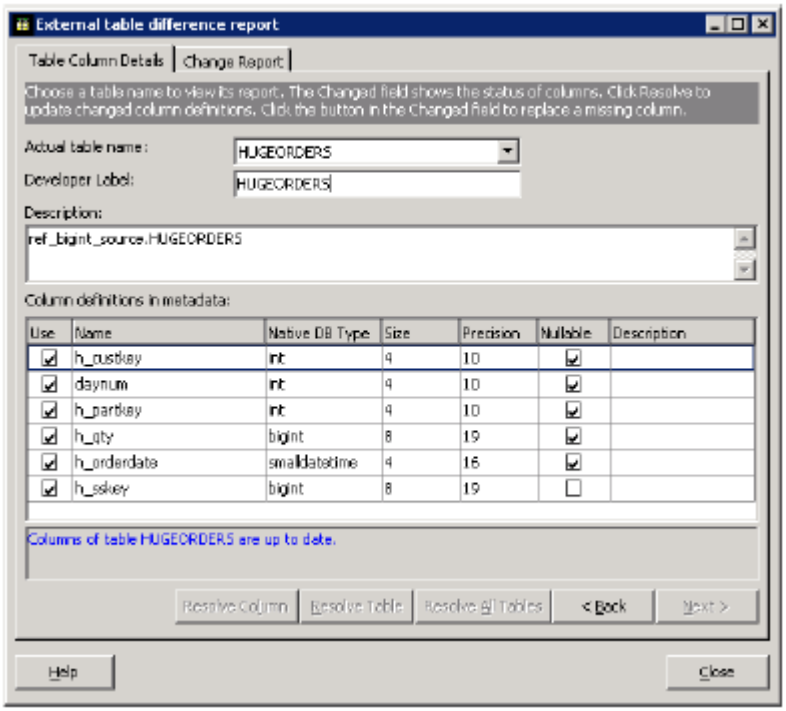


Figure 79: An External Table Difference Report

7 If a table definition has changed in the source database, a message is displayed at the bottom of the dialog box. Changed, new, and missing columns are noted in the **Changed** field of the display area.

8 Click **Resolve Column** to update all selected columns using the definitions in the source database. Click **Resolve Table** to update all columns in the displayed table that are marked as **Changed**. Click **Resolve All Tables** to update changed columns in all tables for which you have generated a report.

9 If the name of a source column has changed, that column is marked as **Missing** in the difference report and you must replace it with the renamed source-table column. To replace a missing column with a source column, click in the **Status** field of the column listing and click the **...** button that appears. In the Replace Missing Entry dialog box that is displayed, select the replacement source-table column and click **Replace**.

Note: If you select a replacement column that is already included in the external table, then the missing column is deleted from the external table and all extraction commands that used the missing column are modified to use the replacement column that you have selected.

Figure 7-10: The Replace Missing Entry Dialog Box

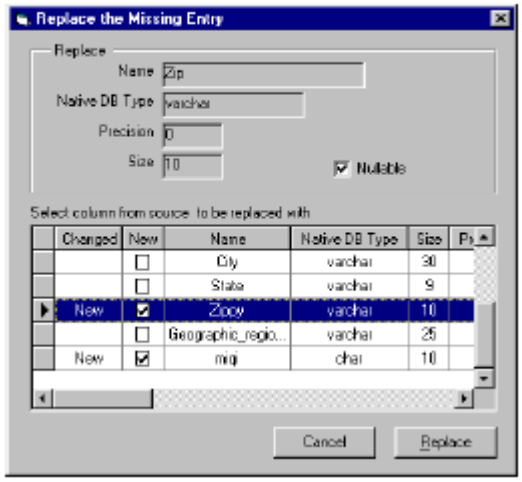


Figure 80: The Replace Missing Entry Dialog Box

10 When you have made all necessary changes, click **Next** to go to the final screen of the wizard.

11 Review the proposed changes that are displayed. Click **Finish** to apply these changes.

Converting External Tables

You can use the Convert External Tables dialog box to generate new external tables or dimension tables in a data store by importing columns from external tables in any data store.

To generate such a table, right click on the Extraction folder of your EpiCenter and select **Convert External Table** to open the Convert External Tables dialog box (see "Figure 81: Convert External Tables Dialog Box: General Tab" on page 221).

Generating a New Table in the Convert External Tables Dialog Box

1 Enter a **Table Name** and **Description** for the new table.

2 Select a Table Type to generate. The available types are **Base dimension table** and **External table in mart**.

Figure 7-11: Convert External Tables Dialog Box: General Tab

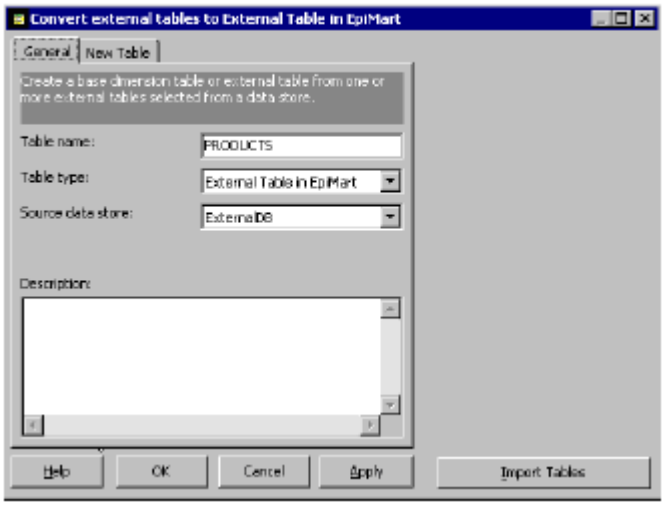


Figure 81: Convert External Tables Dialog Box: General Tab

3 In the **Source Data Store** option, select the data store that contains the tables that you wish to convert.

4 If the tables that you wish to convert is not imported as external tables, click **Import Tables** to use the Import External Tables wizard (see "Importing External Tables" on page 216) to import the desired tables.

Figure 7-12: Convert External Tables Dialog Box: New Table Tab

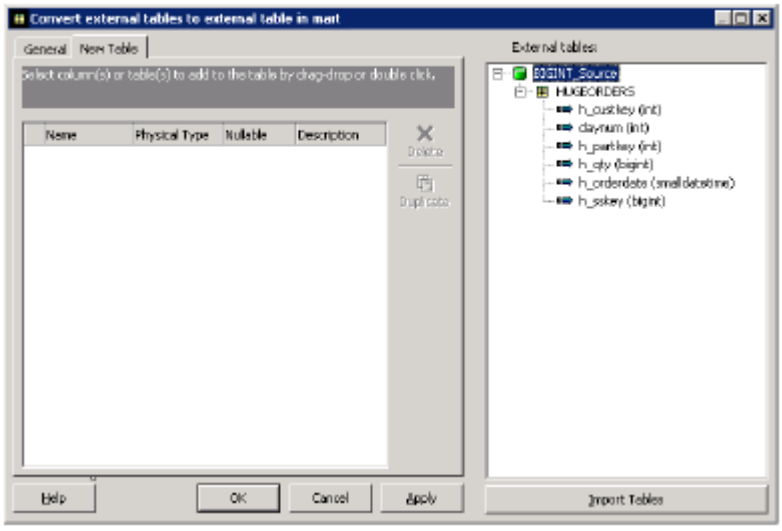


Figure 82: Convert External Tables Dialog Box: New Table Tab

5 You add columns to the table in the **New Table** tab (see "Figure 82: Convert External Tables Dialog Box: New Table Tab" on page 221). Drag columns that you wish to add from the external tables in the right pane to the grid control in the left pane. You can use the grid control to modify or remove columns that you have added.

6 Click **OK** to define the table.

Semantics

8

When you configure an extraction job, you assign a semantic type to each fact or dimension table for which you are extracting data. This semantic type determines how the data from the staging table is to be merged into the data mart table.

A semantic type is a generic merging procedure that resolves potential conflicts within the data. Each semantic type is implemented as a generic SQL program that you can apply to your particular data mart tables. A semantic instance is adapted for the specific fact or dimension table that is being populated. When configuring extraction, you need to select only the semantic type and the destination table. During extraction, an appropriate semantic instance is then automatically created and run. The Infor Campaign Management extraction program, EpiChannel, executes the semantic instance at the appropriate time during an extraction job.

Semantic instances allow you to specify how data updates are to be handled by Infor Campaign Management. The simplest semantic types (Initial Load Dimension and Initial Load Fact) simply overwrite the data mart table with the contents of the staging table. Other semantic types deal intelligently with such issues as backlogs, changes in dimensions over time, and changing fact values when combining new and existing data.

Infor Campaign Management provides a large collection of built-in semantic types, which are described in detail in "Dimension Semantic Types" on page 224, "Running a Fact Roll-Off Extraction Job" on page 177, and "The Measures Folder" on page 64. You can also define additional semantic types, as described in "Defining New Semantic Types" on page 266.

Semantic Instance Extraction Steps

While non-streaming extraction-command extraction steps move data from the source systems (non-Infor Campaign Management data stores) into temporary staging tables in the EpiCenter, semantic instance extraction steps move data from staging tables into fact tables and base dimension tables, where the data can be queried by the end user.

You can define a semantic instance extraction step by using the Semantic Instance dialog box in Admin Manager to assign a semantic type to either a base dimension table or a fact table (see "Semantic Instances" on page 295).

In order to ensure data consistency, data is usually moved into the main data mart tables only by means of a semantic instance. If data is loaded directly into a base dimension or fact table, then a streaming semantic must be applied in order to create the appropriate indexes and ensure data consistency. Therefore, whenever an extraction job extracts data for a base dimension or fact table, the extraction job must also have at least one semantic instance for that table.

You cannot use more than one semantic instance for a single base dimension or fact table, since the semantic types cancel one another out. An exception is that the Count Unjoined semantic type does not actually merge data from the staging table into the EpiMart database. This semantic type has no effect on the base fact tables and is instead useful for its side effects. This special semantic type must be used in conjunction with other semantic types that perform the actual merging of data.

If you are reloading a fact or dimension table in its entirety, you can choose to load the table directly, bypassing the staging table (see "Streaming Extraction" on page 172). When you do this, you must apply a streaming semantic instance to the table after populating it with data in order to ensure data consistency.

Dimension Semantic Types

Dimension semantic types have two purposes: populating base dimension tables and populating dimension mapping tables. If a dimension is mapped (see "Figure 49: Fact Table Dialog Box: Object Props Tab" on page 138), fact tables refer to dimension elements using special Infor Omni-Channel Campaign Management integer dimension keys. Dimension mapping tables are used to convert dimension source system identifier keys (`sskeys`) to these integer dimension keys. Dimension semantic types are also responsible for proper indexing of these two tables.

Infor provides a large collection of pre-defined dimension semantic types that embody sophisticated business logic. You can also define additional dimension semantic types based on any special business logic that you may wish to implement.

Updating UNKNOWN Values

The **UNKNOWN** dimension value (see "UNKNOWN Dimension Values" on page 34) is used to indicate that a value for a dimension column is not known. When you are updating a dimension, you may wish to refrain from performing updates for dimension columns that do not have a known new value.

For example, you may wish to update a customer record with a new address. The dimension element may include the customer’s income, but that information may not be present in the source-system record that is being used to update the address information. In this case, the income field of the extracted records have the `UNKNOWN` value in the staging table. You may not want to replace the income value in the dimension record with the value `UNKNOWN` .

You can configure dimension semantics to skip `UNKNOWN` values when updating dimension records. In the example above, such a semantics update the address in the dimension with the new value, but

it would leave the income value in the dimension unchanged, since that field has the value `UNKNOWN` in the staging table.

Components of Dimension Semantics

Dimension semantics differ with respect to the following behaviors. Note that some combinations of behaviors are not allowed. Conflicting behaviors are marked with flags:

- Stage Duplicates: For each `sskey` , at most one staging-table row with that `sskey` is added to the dimension table. Staging-table rows with the same `sskey` values can be handled in different ways. The available options are:
  - De-dup: Duplicates are removed. For every `sskey` , only the last row with that `sskey` is used.
  - Assert No-dup: The semantic checks for duplicates. If two staging-table rows with the same `sskey` value are found, the semantic halts with an error.
  - Assume No-dup: The semantic does not check for duplicates.

    Note: If you choose a semantic that does not check for duplicate sskeys, you must ensure that no two staging-table rows have the same sskey.

- Base Updates: A semantic can be configured to update dimension rows with the latest staging-table row with a given `sskey`. If a staging-table row has the same `sskey` value as an existing dimension-table row, the following options are available:
  - Any Update: If the staging-table row has different values than the dimension-table row in any column, then the dimension-table row is updated with the new values.
  - Secondary Update: If the staging-table row has different values than the dimension-table row in a secondary column, then the dimension-table row is updated with the new values for that column.
  - No Update: The dimension-table row is not changed.

- New Rows: A semantic can be configured to add a new dimension row with data from a staging-table row under various circumstances. The available options are:
  - Stage Initial: Ignore the current contents of the dimension table and create a new row for every `sskey` in the staging table.
  - Any Update: If a row in the staging table has the same `sskey` as a row in the dimension table and the staging-table row has a different value than the dimension-table row in any column, then add a row to the dimension table with the new data for that `sskey` . Also add a new row to the dimension table for every `sskey` in the staging table that is not found in the dimension table.

    A semantic type with this option cannot be applied to a non-versioned dimension.
  - Primary Update: If a row in the staging table has the same `sskey` as a row in the dimension table and the staging-table row has a different value than the dimension-table row in any primary column, then add a row to the dimension table with the new data for that `sskey` . Also add a new row to the dimension table for every `sskey` in the staging table that is not found in the dimension table. Do not add a new row if all changes are in secondary columns.

    A semantic type with this option cannot be applied to a non-versioned dimension.

    - New sskey: Add a new row to the dimension table for every `sskey` in the staging table that is not found in the dimension table. Do not add new rows for any `sskey` s that are found in the dimension table.
    - Dimension Streaming: New dimension rows are streamed directly into the dimension table (see "Streaming Extraction" on page 172). Validate data consistency and perform indexing.

- Update Criteria: As described in "Updating UNKNOWN Values" on page 224, a dimension column may or may not be considered changed if it contains the `UNKNOWN` value. The available options are:
    - Ignore Unknown: If a staging-table row has the same `sskey` as a dimension-table row, then any columns in the staging-table row with the `UNKNOWN` value are not considered changed.
    - Always: If a staging-table row has the same `sskey` as a dimension-table row, then all columns in the staging-table row with values that are different from those in the dimension-table row are considered changed.

- Rewrite Rows: You can choose whether or not to rewrite dimension rows. The available options for rewriting dimension rows are:
    - No Rewrite: Ignore the contents of the fusion table and do not rewrite dimension rows.
    - Record Fusion: Permanently fuse new records, based on the contents of the fusion table, at the end of semantic execution.

      This option is deprecated as of Version 6.0.0. Use the built-in Latest Dimension Value with Fusion/Fission semantic instead. See "Fusion and Fission" on page 263.
    - Record Fusion/Fission: Fission all previously fused dimension records at the beginning of semantic execution. Fuse records at the end of semantic execution, based on the contents of the fusion table. This allows record fusion to be undone at a later point in time, if necessary.

      This option is deprecated as of Version 6.0.0. Use the built-in Latest Dimension Value with Fusion/Fission semantic instead. See "Fusion and Fission" on page 263.
    - Rewrite All: Permanently rewrite all versions of an SSKEY with the latest value in the staging table.

- Truncate Current: You can choose to truncate the Current table for the dimension after the Next table is successfully populated (see "Swapping Mirrored Tables" on page 192, for more information on Current and Next tables). Since the Infor Campaign Management Server makes use of the Current tables, this option must never be used on an EpiCenter that is being queried by end users.

When you define custom semantics, you can combine these options in a way that reflects your special business logic.

Infor provides a rich set of pre-defined dimension semantic types that are based on the components discussed in this section. The following sections describe the built-in dimension semantic types that are available for every Infor Campaign Management installation.

Built-in Dimension Semantic Types

Note: The Backfeed Dimension semantic is deprecated. To run backfeed dimension semantics, use the built-in Dimension Backfeed Semantic job step. See "..." on page 321 for information.

The following pre-defined semantic types are available for dimensions:

- First Dimension Value . If new values are extracted for an existing dimension element, then those values are ignored. Facts continue to refer to the first values that are extracted for a dimension element.
- Initial Load Dimension . The entire dimension table is reloaded from the staging table, without regard to the current contents of the dimension table. Fact tables that refer to the dimension are invalidated by this semantic.
- Initial Load Dimension, Truncate Current . The entire dimension table is reloaded from the staging table. After dimension data is loaded, the previous copy of the dimension table is truncated. This option must not be used on an EpiCenter that is being queried by end users.
- Latest Dimension Value . If new values are extracted for an existing dimension element, then the dimension table is updated with these new values. All facts refer to the new values.
- Latest Dimension Value, Rewrite All . If new values are extracted for an existing row, then all existing data with the same sskey is updated with the new values as well. This semantic can be used to transition from a Slowly Changing semantic to a Latest Dimension Value semantic.
- Latest Dimension Value, Ignore Unknowns . If new values are extracted for an existing dimension element, then the dimension table is updated with all new values for elements that do not have the `UNKNOWN` value (see "UNKNOWN Dimension Values" on page 34). All facts refer to the new values.
- Latest Dimension Value With Fusion/Fission . For fused dimensions only, if new values are extracted for an existing dimension element, then the dimension table is updated with these new values. All facts refer to the new values. Fission and fusion are performed where appropriate. For a list-producing dimension, this semantic must be used in conjunction with fact fusion semantics. See "Fusion and Fission" on page 263 for details.
- Latest Dimension Value, Preserve Fusion . For fused dimensions only, if new values are extracted for an existing dimension element, then the dimension table is updated with these new values. This semantic must be used on list-producing, fused dimensions whenever fact fusion is not being performed. See "Fusion and Fission" on page 263 for details.
- Slowly Changing Dimensions . If new values are extracted for an existing dimension element, then a new row with these values is added to the dimension table. New facts that refer to this dimension element refer to the new row, but old facts that refer to this dimension element continue to refer to the row containing the data that was current when the fact occurred.
- Slowly Changing Dimensions, Ignore Unknown . If new values, aside from `UNKNOWN` values (see "UNKNOWN Dimension Values" on page 34), are extracted for an existing dimension element, then a new row with these values is added to the dimension table. New facts that refer to this dimension element refer to the new row, but old facts that refer to this dimension element continue to refer to the row containing the data that was current when the fact occurred.
- Slowly Changing/First Dim Value, Ignore Unknown . If a new value, aside from an `UNKNOWN` value, is extracted for a primary column in an existing dimension element, then a new row is added, as with the Slowly Changing Dimensions semantic type. New values in secondary columns are ignored.
- Slowly Changing Primary, Latest Secondary . If a new value is extracted for a primary column in an existing dimension element, then a new row is added, as with the Slowly Changing Dimensions

semantic type. If a new value is extracted for a secondary column in an existing dimension element, then that column is updated with the new value, as with the Latest Dimension Value semantic type.

- Streaming Dimension . The entire dimension table is reloaded directly, bypassing the staging table. See "Streaming Extraction" on page 172.

The built-in dimension semantic types are described in the following sections.

First Dimension Value

The First Dimension Value semantic type ignores any changes to a dimension. Values present when the row was first inserted are preserved forever, regardless of any future changes. This semantic never modifies a row after it is seen.

If you want to use this type if your source data comes from two systems that are not in complete agreement with each other. If one system have Customer #12345 as `David Anderson`, and another system have the same customer as `David Andersen`. Ideally, you have to determine which one was in error and correct it.

In the meantime, you can choose the first value to read and the other to ignore. (This is a good method for avoiding the oscillation problem mentioned on "Slowly Changing Dimensions" on page 231.) If you use the Slowly Changing Dimensions semantic type , then a race between the two source systems for each extraction, and (in the worst case), your dimension values can alter between the two values with every extraction.

Note the following considerations:

- Duplicates are removed from the staging table.
- New `sskey` s from the staging table are inserted in both the dimension table and the mapping table. Rows are added to the mapping tables for mapped dimensions only. Existing `sskey` s are ignored.
- In the EpiChannel log file, the number of new `sskey` rows is reported as the **Inserted** value and the number of rows in the staging table is reported as the **Processed** value.
- First Dimension Value has the same indexing as Slowly Changing Dimensions (see "Built-in Dimension Semantic Types" on page 227).
- The `REAL` column is always equal to the dimension key.

See "Appendix F, “Components of Built-in Semantics,”" on page 533 for a list of the options that are used in the First Dimension Value semantic type.

Initial Load Dimension

The Initial Load Dimension semantic type ignores the current data mart entries. (It is also the fastest semantic type apart from the Streaming Dimension semantic).

Initial Load Dimension loads dimension without regard to any previously existing rows. This semantic type can be used for the initial load of the empty EpiMart database, and also to completely reload a dimension, ignoring existing values. To use any other semantic type you need to empty the existing dimension table before beginning the extraction.

Note the following considerations:

- Duplicates are removed from the staging table.
- The existing dimension and mapping tables are ignored. All `sskey` s are imported directly into the dimension and the mapping tables. Rows are added to the mapping tables for mapped dimensions only.
- In the EpiChannel log file, the number of distinct `sskey` rows is reported as the **Inserted** value and the number of rows in the staging table is reported as the **Processed** value.
- Initial Load Dimension has the same indexing strategy as Slowly Changing Dimensions (see also, "Built-in Dimension Semantic Types" on page 227).
- The `REAL` column is always equal to the dimension key.

See "Appendix F, “Components of Built-in Semantics,”" on page 533 for a list of the options that are used in the Initial Load Dimension semantic type.

Initial Load Dimension, Truncating Current

As with the Initial Load Dimension semantic type, the Initial Load Dimension, Truncate Current semantic type loads data from the staging table to the dimension table, without regard to the current contents of the dimension table. After data is loaded into the Next partition, the copy of the dimension table in the Current partition is truncated. See Also, "Swapping Mirrored Tables" on page 192, for more information on the mirrored Current and Next tables.

Since queries in an active EpiCenter are issued against the Current tables, this semantic must not be used with a system that is being used by an Infor Campaign Management Server.

See "Appendix F, “Components of Built-in Semantics,”" on page 533 for a list of the options that are used in the Initial Load Dimension, Truncate Current semantic type.

Latest Dimension Value

The Latest Dimension Value semantic type updates rows instead of adding versioned rows as in Slowly Changing Dimensions (see "Built-in Dimension Semantic Types" on page 227). It applies changes retroactively to a dimension. Thus, the changes take effect for all historical data, as well as for current and future loads. If there is versioned data, changes are applied to the latest row added to the dimension table.

For example, assume that a sporting goods store has a category historically called `Rollerblades`. Now that they are selling other brands, the store wants to change the category to `In-line Skates`. By using the Latest Dimension Value semantic type, this change can affect all of the historical data because all previous sales of `Rollerblades` are now labeled `In-line Skates`. Consequently, the store can compare year-to-year sales of all in-line skates.

This semantic type removes duplicates from the staging table. Use this semantic type for an implementation that “restates history” when a source dimension table changes.

Note the following considerations:

- New `sskey` s are inserted in both the dimension table and mapping table. Existing `sskey` s with one or more changed dimension columns are updated in place in the dimension table (that is, the same dimension row is used) with the latest values.
- In the EpiChannel log file, the number of new `sskey` rows is reported as the **Inserted** value, the number of changed rows is reported as the **Modified** value, and the number of rows in the staging table is reported as the **Processed** value.
- Latest Dimension Value has the same indexing as Slowly Changing Dimensions (see "Built-in Dimension Semantic Types" on page 227).
- If all updates to a dimension are performed with a Latest Dimension Value semantic, then the `REAL` column is always equal to the dimension key.

See "Appendix F, “Components of Built-in Semantics,”" on page 533 for a list of the options that are used in the Latest Dimension Value semantic type.

Latest Dimension Value, Ignore Unknowns

The Latest Dimension Value, Ignore Unknowns semantic type performs similar updates to those performed by the Latest Dimension Value semantic type. This semantic type differs from Latest Dimension Value in that `UNKNOWN` dimension values (see "UNKNOWN Dimension Values" on page 34) are ignored while performing updates to dimension rows (see also, "Updating UNKNOWN Values" on page 224).

If a dimension-table row is being updated with values from a newer row in the staging table, no changes are specified in a column of the dimension-table row if the staging-table row has the `UNKNOWN` value in that column.

See "Appendix F, “Components of Built-in Semantics,”" on page 533 for a list of the options that are used in the Latest Dimension Value, Ignore Unknowns semantic type.

Latest Dimension Value, Preserve Fusion

The Latest Dimension Value, Preserve Fusion semantic type performs similar updates to those performed by the Latest Dimension Value Semantic type. This semantic must be used on all fused dimensions whenever fact fusion is not being performed (see "Fusion and Fission" on page 263).

Latest Dimension Value, Rewrite All

The Latest Dimension Value, Rewrite All semantic type performs similar updates to those performed by the Latest Dimension Value semantic type. This semantic type differs from Latest Dimension Value in that new values in a row in the staging table cause all dimension table rows with the same sskey to be updated. This semantic can be used to transition from a Slowly-Changing-Dimension-based table to a Latest-Dimension-Value-based table.

See "Appendix F, “Components of Built-in Semantics,”" on page 533 for a list of the options that are used in the Latest Dimension Value, Rewrite All semantic type.

Latest Dimension Value With Fusion/Fission

The Latest Dimension Value With Fusion/Fission semantic type performs similar updates to those performed by the Latest Dimension Value Semantic type. This semantic type differs from Latest Dimension Value in that dimension fission and fusion are performed where appropriate. This semantic must only be used in conjunction with a fact fusion semantic (see "Fusion and Fission" on page 263).

Slowly Changing Dimensions

A Slowly Changing Dimension is a dimension in which the attributes or hierarchy of the dimension can change over time, but historical data is not restated. Two examples follow:

- A national sporting goods chain has stores divided into three regions. On January 1, 1998, the chain reorganizes its regions, moving Denver from the Central to the Western region. Their 1998 sales forecasts take into account that the Denver store is in the Western region, but they do not want to recalculate forecasts and actuals from previous years. Using the Slowly Changing Dimensions semantic type, sales for Denver can be aggregated up to the Central region through 1997. Beginning January 1, 1998, Denver sales are applied to the Western Region.
- On May 7, 1999, one of this store's customers changes careers and doubles her income. Since the store analyzes customer purchases based on demographics, all of this customer's new purchases must be analyzed with her new demographic data. However, her previous purchases must still be analyzed based on her old demographic data. Using the slowly changing dimensions semantic type, all of this customer's purchases are recorded based on her demographics at the time of the purchase.

The built-in Slowly Changing Dimensions semantic type accomplishes the following logic:

- If an `sskey` appears in more than one row of the staging table, then duplicates are eliminated by following a "last in wins" rule. The last row is determined by the special column called ikey , which is created by EpiChannel and is automatically incremented during normal extractions. The highest `ikey` row for a given `sskey` is accepted.
- New rows are created by searching through the dimension-staging table for new `sskey` values, or `sskey` values that have one or more dimension column changes from the last known values. Each of these cases creates a new row in the dimension table. The mapping row for that `sskey` points to the latest dimension row with that `sskey` value.
- In the EpiChannel log file, the number of new `sskey` rows is reported as the **Inserted** value, the number of changed rows is reported as the **Modified** value, and the number of rows in the staging table is reported as the **Processed** value.
- The key column of the dimension table becomes the primary-key index. By default, each dimension column is also Indexed (non-uniquely). In a mapped dimension, the mapping table is indexed (primary key) on `sskey` . You can use the Dimension Column dialog box in Admin Manager to disable the indexing of individual columns.
- The column dimension `_key_REAL` (where dimension is the dimension table name) is set to the first key value for each `sskey` . In other words, when a new `sskey` is discovered, the `REAL` key is set to the new dimension key value. Subsequent dimension rows for this `sskey` are retained to the original `REAL` key value.

Note the following considerations:

- Do not allow dimension column values to oscillate unpredictably. (See "Built-in Dimension Semantic Types" on page 227 for more information.) In particular, do not rely on `ikey` filtering of duplicate `sskey` values if two or more rows during a single extraction can have different values for one or more dimension columns. The reason is that a new row is created in the dimension table for every extraction for which a change is recorded. Consequently, two values can “compete” with each other, forcing an unending sequence of row creation in the dimension table.
- Rows can only be removed from dimension tables after they are extracted with an explicit delete or truncation, or through use of the Initial Load Dimension semantic.
- The `UNKNOWN` dimension row always has a key value of `1`. Since the `key` value is constant, the `UNKNOWN` row does not have an entry in the mapping table.
- You cannot use a Slowly Changing Dimensions semantic instance on a non-versioned dimension.

See "Appendix F, “Components of Built-in Semantics,”" on page 533 for a list of the options that are used in the Slowly Changing Dimensions semantic type.

Note: State-like facts are not limited to a moment in time, but are relevant until negated. In the absence of negating data, the last state carries forward. However, when a dimension row changes in a slowly changing dimension, the new dimension row is not linked to the existing state unless a new fact row is extracted to force this linkage.

For example, if you populate a fact table with information about which individuals belong to which groups, fact rows are written that link a particular indiv_key to a group_key. If an individual changes, the Slowly Changing Dimension semantic causes a new indiv_key to be created for this individual. This key is now the “latest” for this indiv_key_REAL. However, unless the extraction job also writes a new fact row that links the new version of the indiv_key to the same group, the fact table still references the old data information for the individual in this group. The new individual information is not tied to the group.

Slowly Changing Dimensions, Ignore Unknowns

The Slowly Changing Dimensions, Ignore Unknowns semantic type performs similar updates to those performed by the Slowly Changing Dimensions Semantic type. This semantic type differs from Slowly Changing Dimensions in that `UNKNOWN` dimension values (see "UNKNOWN Dimension Values" on page 34) are not used to update dimension-table rows (see "Updating UNKNOWN Values" on page 224).

If a staging-table row is identical to the most recent dimension row with the same `sskey` except for columns in the staging-table row that contain `UNKNOWN` values, then no new row is added to the dimension for that staging-table row. If a staging-table row differs from the most recent dimension row with the same `sskey` in some column that contains a value different from `UNKNOWN` in the staging table, then a new row is added to the dimension table. This new row contains all non-`UNKNOWN` values from the staging-table row. `UNKNOWN` values from the staging-table row are replaced with the values found in the most recent dimension row with the same sskey.

See "Appendix F, “Components of Built-in Semantics,”" on page 533 for a list of the options that are used in the Slowly Changing Dimensions, Ignore Unknowns semantic type.

Slowly Changing/First Dim value, Ignore Unknown

The Slowly Changing/First Dim Value, Ignore Unknown semantic type adds a new row whenever new data is extracted for a primary column of an existing dimension element. This new row contains the newly extracted values for all primary dimension columns and the existing values (from the dimension table) for all secondary dimension columns. `UNKNOWN` dimension values (see "UNKNOWN Dimension Values" on page 34) are not used to update dimension-table rows (see "Updating UNKNOWN Values" on page 224) and changes to secondary columns are ignored.

See "Appendix F, “Components of Built-in Semantics,”" on page 533 for a list of the options that are used in the Slowly Changing/First Dim Value, Ignore Unknown semantic type.

Slowly Changing Primary, Latest Secondary

The Slowly Changing Primary, Latest Secondary semantic type is like the Slowly Changing Dimensions semantic type for changes in Primary dimension columns and like the Latest Dimension Value semantic type for changes in Secondary column values (see "Primary and Secondary Dimension Columns" on page 34 for more information on primary and secondary columns).

If a row with an `sskey` that already appears in the base dimension table is extracted to the staging table, then the following occurs:

- Duplicates are filtered as with the Slowly Changing Dimensions and Latest Dimension Value semantic types.
- If the staging-table row contains a value in a primary column that is different from the value in that column in the latest row of the base dimension table with that `sskey` , then a new row is added, as with the Slowly Changing Dimensions semantic type.
- If the staging-table row contains a value in a secondary column that is different from the value in that column in the latest row of the base dimension table with that `sskey` , then the latest dimension row with that `sskey` is updated with the new value. This update occurs even if a new dimension row is added as a result of a change in a primary column.

See "Appendix F, “Components of Built-in Semantics,”" on page 533 for a list of the options that are used in the Slowly Changing Primary, Latest Secondary semantic type.

Streaming Dimension

As described in "Streaming Extraction" on page 172, if you are reloading a dimension, you can stream data directly into the dimension table, bypassing the staging table. The Streaming Dimension semantic type creates indexes and performs consistency checks on a dimension that are loaded by streaming extraction. You must apply a Streaming Dimension semantic after populating a dimension table with streaming extraction.

See "Appendix F, “Components of Built-in Semantics,”" on page 533 for a list of the options that are used in the Streaming Dimension semantic type.

Note: When you use a Streaming Dimension semantic instance in an extraction job, the Force rebuild of target table option in the Extraction Steps tab of the Job dialog box is ignored during execution of the semantic.

Dimension Data Deletion

You can configure a dimension semantic to delete existing data records. For example, if a customer asks to be removed from your Campaign Management database, you can use the dimension data deletion feature to completely remove that customer's dimension records.

To delete dimension data, run an extraction job that includes the following:

- An extraction command that is configured to populate the dimension delete table for the dimension from which you are deleting data. This extraction command must extract the sskey values for the dimension records that you wish to delete.
- A semantic instance for the dimension from which you are deleting data that has the value of the **Dimension Data Delete** parameter set to 1.

After you run this job, the current mirror dimension is no longer included in the deleted dimension records, though the non-current mirror still includes such data. The next job that runs a semantic against the dimension is then deletes the dimension records from the other mirror, after which the dimension records are fully removed. Note that the semantic in this second job does not need to have the **Dimension Data Delete** parameter set to 1. The **Dimension Data Delete** parameter in the semantic only needs to be set to 1 when you are extracting new data to the dimension delete table.

If you have deleted dimension data, fact semantics that run against fact tables that join to the dimension updates any fact records that previously referred to the deleted dimension records. These fact records are updated to refer to the UNKNOWN dimension row. Note that it takes two jobs with fact semantics run against a given fact table to update both mirrors of that fact.

After data is deleted from both dimension mirrors and both mirrors of all facts that join to the dimensions are updated, the deletion key data is automatically removed from the dimension delete tables.

Comparison of Default Dimension Semantic Types

This section illustrates the differences between the default Infor Campaign Management semantic types by means of examples from a fictitious data mart containing customer, product, and order data. The "Initial Customer Staging Table" on page 234 and the "Initial Product Staging Table" on page 235 below show a set of rows that appears in a pair of dimension staging tables after an initial extraction.

Table 28: Initial Customer Staging Table

ikey	sskey	Customer_Name	Age	Date
1	bobb	Bob Bobster	47	4/23/99
2	lobt	Lobby Tableman	15	4/7/99
3	betk	Betty Kezer	31	4/15/99

Table 29: Initial Product Staging Table

ikey	sskey	Product_Name	Date
1	cool	Cool Stuff 2	4/30/99
2	wow	Wowzer Pro	4/30/99
3	neat	Neato Lite	4/30/99

The staging table rows are numbered sequentially by the value in the `ikey` column.

The "Initial Customer Dimension Table" on page 235 and "Initial Product Dimension Table" on page 235 show how these tables are loaded into the data mart with an Initial Load Dimension semantic.

Table 30: Initial Customer Dimension Table

key	Customer_Name	Age	Date	key_REAL
1	UNKNOWN	0	1/1/1900	1
2	Bob Bobster	47	4/23/99	2
3	Lobby Tableman	15	4/7/99	3
4	Betty Kezer	31	4/15/99	4

Table 31: Initial Product Dimension Table

key	Product_Name	Date	key_REAL
1	UNKNOWN	1/1/1900	1
2	Cool Stuff 2	4/30/99	2
3	Wowzer Pro	4/30/99	3
4	Neato Lite	4/30/99	4

The `key_REAL` column records the first row containing an entry for that `sskey` . In an initial load, the `key_REAL` value is always equal to the `key` value.

Each dimension table also has an `UNKNOWN` row, which always has a `key` value of 1 .

The data mart also has mapping tables that map `sskey` s to dimension table **keys**. For example, the Customer mapping table looks like this sample table:

Table 32: Initial Customer Mapping Table

sskey	key
bobb	2
lobt	3
betk	4

In a subsequent extraction, you can load the new data shown in the table below into the Customer staging table.

Table 33: Second Customer Staging Table

ikey	sskey	Customer_Name	Age	Date
1	jrd	J.R. Dobbs	23	5/10/99
2	lobt	Lobby Tableman	16	5/5/99
3	jrd	J.R. Dobbs	20	5/2/99
4	bobb	Bob Bobster	47	4/23/99

This data can be merged into the dimension tables in different ways, depending on your choice of dimension semantic.

Note that all dimension semantics ignore the first row, since there is another staging table row with the same `sskey` and a higher `ikey` value. All semantics also ignore the last row, since it is a duplicate of a row that is already in the dimension table (row 2).

Slowly Changing Dimensions

If the new data is merged with a Slowly Changing Dimensions semantic, then the revised dimension table looks like "Table 34: Customer Dimension Table After Slowly Changing Dimensions Semantic" on page 236.

Table 34: Customer Dimension Table After Slowly Changing Dimensions Semantic

key	Customer_Name	Age	Date	key_REAL
1	UNKNOWN	0	1/1/1900	1
2	Bob Bobster	47	4/23/99	2
3	Lobby Tableman	15	4/7/99	3
4	Betty Kezer	31	4/15/99	4
5	Lobby Tableman	16	5/5/99	3
6	J.R. Dobbs	20	5/2/99	6

The new data for Lobby Tableman is appended to the dimension table. The `key_REAL` value of the new row points to the first row with that `sskey` , which is row 3.

The revised mapping table looks like "Table 35: Customer Mapping Table After Slowly Changing Dimensions Semantic" on page 237.

Table 35: Customer Mapping Table After Slowly Changing Dimensions Semantic

sskey	key
bobb	2
lobt	5
betk	4
jrd	6

The mapping table now associates the `sskey` for Lobby Tableman with the new row, and all new facts that refer to customer `sskey lobt` can have a Customer foreign key of 5. However, previously extracted facts that referred to `sskey lobt` continue to have a Customer foreign key of 3.

First Dimension Value

If the new data is merged with a First Dimension Value semantic, then the revised dimension table looks like "Table 36: Customer Dimension Table After First Dimension Value Semantic" on page 237.

Table 36: Customer Dimension Table After First Dimension Value Semantic

key	Customer_Name	Age	Date	key_REAL
1	UNKNOWN	0	1/1/1900	1
2	Bob Bobster	47	4/23/99	2
3	Lobby Tableman	15	4/7/99	3
4	Betty Kezer	31	4/15/99	4
5	J.R. Dobbs	20	5/2/99	5

The data for the new customer is appended to the table, and the revised data for Lobby Tableman is discarded.

The revised mapping table looks like "Table 37: Customer Mapping Table After First Dimension Value Semantic" on page 237.

Table 37: Customer Mapping Table After First Dimension Value Semantic

sskey	key
bobb	2
lobt	3
betk	4
jrd	5

Latest Dimension Value

If the new data is merged with a Latest Dimension Value semantic, then the revised dimension table looks like "Table 38: Customer Dimension Table After Latest Dimension Value Semantic" on page 238.

Table 38: Customer Dimension Table After Latest Dimension Value Semantic

key	Customer_Name	Age	Date	key_REAL
1	UNKNOWN	0	1/1/1900	1
2	Bob Bobster	47	4/23/99	2
3	Lobby Tableman	16	5/5/99	3
4	Betty Kezer	31	4/15/99	4
5	J.R. Dobbs	20	5/2/99	5

The data for the new customer is appended to the table, and the revised data for Lobby Tableman is used to update row 3. The revised mapping table looks like "Table 39: Customer Mapping Table After Latest Dimension Value Semantic" on page 238.

Table 39: Customer Mapping Table After Latest Dimension Value Semantic

sskey	key
bobb	2
lobt	3
betk	4
jrd	5

Fact Semantic Types

Fact semantic types are used to populate and index fact tables. Infor provides a large collection of pre-defined fact semantic types that embody sophisticated business logic. You can also define additional fact semantic types based on special business logic that you implement.

Fact semantics operate on different types of facts:

- Transactional

  These facts indicate some discrete event, such as a shipment or a web site visit.
- State-like

  These facts describe the current state of some business quantity, such as an inventory level or the state of an open order.
- Pipelined

These facts are state-like facts that are tracked through several life cycle phases (for example, sales opportunity or support call facts).

Note: Transactional fact rows with all zero facts are discarded by all semantics except Initial Load and Streaming semantics.

Components of Fact Semantics

Fact semantics differ with respect to the following behaviors. Note that some combinations of behaviors are not allowed. Conflicting behaviors are marked with flags:

- Row Type: Staging-table rows can be treated as transactions or as reports of a state. The available options are:
  - Transactional: All staging-table rows are transactional.
  - Statelike: All staging-table rows are state-like.
  - Transactional/Statelike: Staging-table rows can be transactional or state-like. The value in the `process_key` column determines whether a given row is transactional or state-like.

  Note: If a table has both transactional and statelike data, then you must not use the same sskey to refer to both types of facts.

  - Streaming Facts: New fact rows are streamed directly into the fact table (see "Streaming Extraction" on page 172). Validate data consistency and perform indexing.
  - Pipelined . All staging-table rows are state-like data used for a pipelined fact. Pipelined facts are tracked as they move through several life cycle phases (for example, sales opportunity or support call facts) .

    Note: If a fact table contains a column of type TINYINT, then any semantic with a row type of Statelike, Transactional/Statelike, or Pipelined cannot be used with that fact table.

  Note: If a fact table contains a column of type TINYINT, then any semantic with a row type of Statelike, Transactional/Statelike, or Pipelined cannot be used with that fact table.

- Force Close: If an `sskey` is found in the fact table but not in the staging table, that `sskey` can be “forced closed” (see "Built-in Fact Semantic Types Details" on page 243"Built-in Fact Semantic Types Details" on page 243). This option does nothing if the staging table is not populated.

  Note: If a fact table contains a column of type TINYINT, then any semantic with the Force Close option set to Yes cannot be used with that fact table.

- Use Current:

  The new version of the fact table can be populated entirely from the staging table, or it can be populated by merging the data in the staging table with the data in the current version of the fact table.

- Same date in stage:

  Staging-table rows with the same `sskey` values and the same `date_key` values can be merged in different ways. The available options are:

  - All: All rows are used for transactional facts. For each combination of `sskey` and `date_key` values, the last row of the staging table that has those values is used for state-like facts.
  - Error . The semantic halts with an error.
  - First . For each combination of `sskey` and `date_key` values, the first row of the staging table that has those values is used.

- Later date in stage:

  Staging-table rows with the same `sskey` values and different `date_key` values can be merged in different ways. The available options for the remaining rows are:

  - All: All rows are used.
  - Error: The semantic halts with an error.
  - First: For every `sskey` , the single earliest legal date is selected.

- Truncate current fact .

  You can choose to truncate the Current table for the fact after the Next table is successfully populated (see "Swapping Mirrored Tables" on page 192, for more information on Current and Next tables). Since the Infor Campaign Management Server specifies the use of Current tables, this option must never be used on an EpiCenter that is being queried by end users.

If staging-table data is merged with current fact-table data, then all current fact-table rows with `sskey` values that are not found in the staging table are used in the new version of the fact table. New rows are added to the fact table for all `sskey` values that are found in the staging table but not in the current fact table.

For `sskey` values that are found in both the current fact table and the staging table, staging-table rows can be merged in different ways based on `date_key` values. If rows with duplicate `sskey` values are removed from the staging table, as described above, then those rows are not merged into the fact table. The remaining rows in the staging table are merged with the current fact table to produce the new fact table.

For a staging-table row with an `sskey` that is also found in the current fact table, the `date_key` value determines whether the staging-table row is added to the new fact table. The `date_key` value of the staging-table row is compared to the highest `date_key` value that is found for that `sskey` in the fact table, as follows:

- Same date in stage/current: If the value of `date_key` in the staging table is the same as the highest `date_key` value for the `sskey` in the fact table, the following options are available:
  - Both: For state-like data, the staging-table row is added to the fact table. For transactional data, the staging-table row is not added to the fact table.
  - Current: The staging-table row is not added to the fact table.
  - Error: The semantic halts with an error.

- Later date in stage/current: If the value of `date_key` in the staging table is later than the highest `date_key` value for the `sskey` in the fact table, the following options are available:
  - Both: The staging-table row is added to the fact table.
  - Current: The staging-table row is not added to the fact table.

    - Error: The semantic halts with an error.

- Earlier date in stage/current: If the value of `date_key` in the staging table is earlier than the highest `date_key` value for the `sskey` in the fact table, the following options are available:
    - Both: The staging-table row is added to the fact table if it is transactional. The staging-table row is not added to the fact table if it is state-like.
    - Current: The staging-table row is not added to the fact table.
    - Error: The semantic halts with an error.

When you define custom semantics, you can combine these options in a way that reflects your special business logic.

The built-in fact semantic types are described in the following sections.

Built-in Fact Semantic Types Summary List

Note: The Backfeed Fact Rolloff semantic type is deprecated. To run backfeed fact rolloff semantics, use the built-in BackfeedRolloff job template. See "The Backfeed Roll-Off Job" on page 324 for information.

The following pre-defined semantic types are available for facts:

Backfeed Fact	Produces a fact backfeed table after running campaigns. This backfeed data can then be made available to Infor Campaign Management users by extracting it into data mart tables. You can also use the Backfeed Fact job template to perform Backfeed semantics on all list-producing-dimension-based fact tables at once. See "The Backfeed Job" on page 321 for information.
First/Last Fact	Only the first and last occurrence of a fact is recorded in the fact table. This is an initial semantic.
Ignore New Fact Data	Ignores the contents of the staging table and performs the appropriate rebuild/revise of the base table with no new data (performing fact compression or fusion/fission updates if those options are selected).
Initial Load Fact	The entire fact table is reloaded from the staging table, without regard to the current contents of the fact table. All facts in the staging table are assumed to be transactional.
Initial Load Fact, Truncate Current	The entire fact table is reloaded from the staging table. After fact data is loaded, the previous copy of the fact table is truncated. This option should

	never be used on an EpiCenter that is being queried by end users.
Initial Load Statelike	The entire fact table is reloaded from the staging table, without regard to the current contents of the fact table. All facts in the staging table are assumed to be state-like.
Initial Load Transactional/Statelike	The entire fact table is reloaded from the staging table, without regard to the current contents of the fact table. Staging-table data is added as with the Transactional/Statelike semantic.
Initial Load Transactional/Statelike/Force Close	The entire fact table is reloaded from the staging table, without regard to the current contents of the fact table. Staging-table data is added as with the Transactional/Statelike/Force Close semantic.
Pipelined	Facts are tracked as they move through several life cycle phases.
Pipelined/Forceclose	Facts are tracked as they move through several life cycle phases. Any open booking that is in the fact table but not in the staging table is automatically closed.
Reload Date Fact	All facts that occurred after the earliest date in the staging table are reloaded from the staging table, as with the Initial Load Fact semantic.
Statelike	State-like data is transformed into transactional data and added to the fact table. The staging table is assumed to contain only state-like data.
Statelike/Error on Stage Duplicates	State-like data is transformed into transactional data and added to the fact table. The staging table is assumed to contain only state-like data. The semantic fails if the staging table contains more than one row with a single sskey.
Statelike/Single Delta	State-like data is transformed into transactional data and added to the fact table. For each sskey, only the earliest legal value in the staging table is used. The staging table is assumed to contain only state-like data.
Streaming Fact	The entire fact table is reloaded directly, bypassing the staging table. See "Streaming Extraction" on page 172.
Transactional	Transactional data is added to the fact table. The staging table is assumed to contain only transactional data.

Transactional Incremental Streaming	Fact information is inserted directly, bypassing the staging table. See "Streaming Extraction" on page 172.
Transactional/Dedup Stage	At most one row of transactional data is added to the fact table for each sskey in the staging table. For each sskey, only the earliest date value in the staging table is used, and only the first row with this date is used. The staging table is assumed to contain only transactional data.
Transactional/Statelike	Transactional data is treated as in the Transactional semantic type. State-like data is transformed into transactional data and then merged into the fact table.
Transactional/Statelike/Force Close	This semantic is similar to the Transactional/State-Like semantic, except that all open bookings must be found in the staging table. Any open booking that is in the fact table but not in the staging table is automatically closed.
Undo Fact	This semantic removes information from backfeed fact tables about campaigns that are undone. See "Campaign Undo" on page 201 for information.

There is also one fact semantic that specifies no changes to data mart data, and that is used for informational purposes:

Count Unjoined	This semantic informs you of the number of rows that are transformed by an outer join to refer to UNKNOWN dimension rows. No actual updates are performed.

Built-in Fact Semantic Types Details

The following sections describe these fact semantic types in detail.

Count Unjoined

Count Unjoined informs you of the number of rows that are transformed by an outer join when facts are loaded from a staging table to the main fact tables. In order to prevent the loss of rows, outer joins are used to map fact-staging tables to mapped dimension tables. Any unmatched dimension keys in fact-staging tables are automatically set to refer to the special `UNKNOWN` dimension row. The Count Unjoined semantic counts the number of rows that are transformed in this way and displays the result in the console window.

Note: Only unjoined keys in the staging table are counted. The Count Unjoined semantic does not count unjoined keys in streaming extraction steps or keys that are remapped to the UNKNOWN dimension row in EpiChannel memory before being inserted into the staging table.

Keys can be remapped in EpiChannel memory if a dimension has the Allow EpiChannel to Map SSKeys to Keys in Memory option selected. (See "Base Dimension Tables" on page 100.)

A `Count Unjoined` semantic does not make changes to any data mart tables.

The `Count Unjoined` semantic type uses a custom template that is not based on the options described in "Components of Fact Semantics" on page 239.

First/ Last Fact

The First/Last Fact initial semantic type keeps track of the first and last fact values for an `sskey`.

For example, if a store wish to keep track of only the first and last purchases that a customer has specified. This can be done with the First/Last Fact semantic type by using the customer ID as the `sskey`.

The First/Last Fact semantic type does not use the transaction type values in the staging table. Instead, it uses the `FIRST` and `LAST` transaction types. By default, the values for `FIRST` and `LAST` are `120` and `121`, respectively. The First/Last Fact semantic type performs the following actions:

- If two or more staging table rows have the same `sskey` and date values, only the last of these rows is used.
- For every `sskey` in the staging table, the staging table row with that `sskey` and the earliest date value is found.
  - If the fact table does not contain a row with this `sskey`, then the staging table row is added to the fact table with a `transtype_key` value of `FIRST`.
  - If the fact table contains a row with this `sskey`, a `transtype_key` value of `FIRST`, and a later date, then the values in the fact table row are replaced with the values in the staging table row. The value of `transtype_key` is set to `FIRST`.
  - If the fact table contains a row with this `sskey` and an equal or earlier date, then the staging table row is ignored.
- For every `sskey` in the staging table, the staging table row with that `sskey` and the latest date value is found.
  - If the fact table does not contain a row with this `sskey`, then the staging table row is added to the fact table with a `transtype_key` value of `LAST`.
  - If the fact table contains a row with this `sskey`, a `transtype_key` value of `LAST`, and an earlier date, then the values in the fact table row are replaced with the values in the staging table row. The value of `transtype_key` is set to `LAST`.
  - If the fact table contains a row with this `sskey` and an equal or greater date, then the staging table row is ignored.

Note the following considerations:

- The FIRST and LAST transaction types, which are required for this semantic type, are not installed by default. To use these this semantic, you must first install the auxiliary transaction types metadata, which is located in:

  <Infor Campaign Management install directory>\ConfigFiles\Metadata\en\Transtypes_BBB.mdb

  -or the full default path:

  <Infor Campaign Management install directory>\Infor\Campaign Management\10.1.0\ConfigFiles\Metadata\en\Transtypes_BBB.mdb

- The First/Last Fact semantic type ignores the `transtype_key` values in the staging table.
- The First/Last Fact semantic type always adds two fact table rows for every new `sskey` in the staging table. If the new `sskey` only appears in one staging table row, than these two fact table rows are identical except for the value of `transtype_key` .

The First/Last Fact semantic type uses a custom template that is not based on the options described in "Components of Fact Semantics" on page 239.

Ignore New Fact Data

Use this semantic when you need to run a fact semantic without new fact data. The following list provides some reasons for running a fact semantic without new data:

- fact compression
- fact fusion
- prompting MomentumBuilder to specify changes to tables or indexes related to a fact. MomBuilder only works on facts and dimensions that have semantics run against them in the same job.
- equalizing fact mirrors in preparation for fact rolloff.

Initial Load Fact

Similar to the Initial Load Dimension, the Initial Load Fact semantic type is the fastest way to load a fact table from a staging table. It also has the special property that the current tables are ignored so that this semantic type can reload an already-populated fact table. All data in the staging table is assumed to be transactional.

All existing rows in the fact table are lost. For this reason, Initial Load Fact is usually used during development when an installation is verifying whether a first extraction yields correct data.

Note: For first time extractions, you can use this semantic template in place of all other fact semantic types that load data when each sskey is being loaded into the staging table only once. This is because upon first extract, when an sskey is loaded only once, the special logic of the other semantic types (such as delta inference during Statelike) does not apply. If the first extraction does require multiple records per sskey, such as loading inventory values using Statelike, then you must use one of the other Initial Load semantic types.

See "Appendix F, “Components of Built-in Semantics,”" on page 533 for a list of the options that are used in the Initial Load Fact semantic type.

Initial Load Fact, Truncate Current

As with the Initial Load Fact semantic type, the Initial Load Fact, Truncate Current semantic type loads data from the staging table to the fact table, without regard to the current contents of the fact table. After data is loaded into the Next partition, the copy of the fact table in the Current partition is truncated. See "Swapping Mirrored Tables" on page 192, for more information on the mirrored Current and Next tables.

Since queries in an active EpiCenter are issued against the Current tables, this semantic must never be used with a system that is being used by an Infor Campaign Management Server.

"Appendix F, “Components of Built-in Semantics,”" on page 533 lists the options that are used in the Initial Load Fact, Truncate Current semantic type.

Initial Load Statelike

As with the Initial Load Fact semantic type, the Initial Load Statelike semantic type loads data from the staging table to the fact table, without regard to the current contents of the fact table. All data in the staging table is treated as state-like and added to the fact table using the same logic as the Statelike semantic type.

See "Appendix F, “Components of Built-in Semantics,”" on page 533 for a list of the options that are used in the Initial Load Statelike semantic type.

Initial Load Transactional/Statelike

As with the Initial Load Fact semantic type, the Initial Load Transactional/ Statelike semantic type loads data from the staging table to the fact table, without regard to the current contents of the fact table. As with the Transactional/Statelike semantic type, data in the staging table is treated as transactional or state-like depending on the value in the `process_key` column. Data is added to the fact table using the same logic as the Transactional/Statelike semantic type.

See "Appendix F, “Components of Built-in Semantics,”" on page 533 for a list of the options that are used in the Initial Load Transactional/Statelike semantic type.

Initial Load Transactional/Statelike/Force Close

As with the Initial Load Fact semantic type, the Initial Load Transactional/ Statelike/Force Close semantic type loads data from the staging table to the fact table, without regard to the current contents of the fact table. As with the Transactional/Statelike semantic type, data in the staging table is treated as transactional or state-like depending on the value in the `process_key` column. Data is added to the fact table using the same logic as the Transactional/Statelike/Force Close semantic type. If the staging table is not populated, the Force Close option has no effect.

See "Appendix F, “Components of Built-in Semantics,”" on page 533 for a list of the options that are used in the Initial Load Transactional/Statelike/Force Close semantic type.

Pipelined

Use the Pipelined semantic type for facts that can exist in several different life-cycle phases, called pipeline states; for example, sales opportunities or support calls facts.

The `transtype_key` field represents the pipeline. Before using this semantic type, you must define the pipeline stages and numbers.

Pipelined generates these transactions:

- When an `sskey` first enters some pipeline stage (`transtype_key`), a positive fact row is created with that `transtype_key` .
- If that `sskey` subsequently moves backwards in the pipeline (that is, if the same `sskey` appears with a smaller transtype value), then in addition to the new row created by the previous step for the new pipeline stage, a negative “de-booking” transaction is created with the old `transtype_key` minus 1. (This is a loss transaction.)
- If that `sskey` subsequently moves forwards in the pipeline (that is, if the same `sskey` appears with a larger transtype value), then in addition to the new row created in step 1 on page 247 for the new pipeline stage, a negative “de-booking” transaction is created with the old `transtype_key` plus 1. (This is a shipment transaction.)

  Note: The transtype values for all pipeline stages must be multiples of 3. The +1 and -1 transtype values used in step 2 on page 247 and step 3 on page 247 above are created automatically from these values.

For example, assume that the pipeline consists of the following steps: Prospect, Lead, Quote, and Order. You can set up `transtype_key` fields for this pipeline as follows: 30003= Prospect, 30006 = Lead, 30009 = Quote, and 30012 = Order. The semantic type creates its own transactions for 30002, 30004, 30005, 30007, 30008, 30010, 30011, and 30013, which correspond to the forward and backward movements from the various pipeline stages.

To define measures that extract information such as Number of New Leads, you must define transaction types with these new names and keys using the Configuration Dialog box in Admin Manager (see "The Configuration Dialog Box" on page 147).

See "Appendix F, “Components of Built-in Semantics,”" on page 517for a list of the options that are used in the Pipelined semantic type.

Pipelined/ Force Close

The Pipelined/Force Close semantic type is equivalent to the Pipelined semantic type with the “force-close” logic described in "Built-in Fact Semantic Types Details" on page 243 (although it does not filter on transaction type). When a force-close transaction is required for a particular sskey, the Pipelined/Force Close semantic type creates a single debooking with the latest used date and the transtype value of the last open stage. If the staging table is not populated, the Force Close option has no effect.

See "Appendix F, “Components of Built-in Semantics,”" on page 533 for a list of the options that are used in the Pipelined/Force Close semantic type.

Reload Date Fact

The Reload Date Fact initial semantic type is a hybrid of the Transactional and Initial Load Fact semantic types. Use it to reload a subset of an existing table in which the reload point is separable by date only.

Reload Date Fact first determines the minimum date that occurs in the fact-staging table. It then copies from the current fact table to the new fact table all existing EpiMart data that occurs with a date key prior to that minimum date.

The fact-staging table is then used to hold all subsequent dates. Thus, a complete load of data must be placed in the fact-staging data from the minimum date forward. When loading the fact-staging table (with an extraction command), use a filter (or a WHERE clause) based on a particular date.

The Reload Date Fact semantic type uses a custom template that is not based on the options described in "Components of Fact Semantics" on page 239.

Note: Statelike, First/Last and Pipelined facts cannot be reloaded with this semantic. This semantic only works with transactional data.

Statelike

The Statelike semantic type allows changes to data in the EpiMart database. Records in the staging table are treated as states (such as orders or inventory levels) that can be differenced between extractions (a discussion of differencing follows).

The built-in Statelike semantic type first applies the following logic:

- For every staging table row, if the `date_key` value for that staging table row is less than the last `date_key` value for that `sskey` in the fact base table, then the staging table row is discarded. In other words, a state (such as an order) can only be modified on its last reported date, or at some time further into the future.
- If two or more rows in the staging table have the same `sskey` and `date_key` values, then only the row with the highest `ikey` value is used for that combination of `sskey` and `date_key` . Other rows are discarded. This is the same type of filtering that occurs in dimension semantics such as Slowly Changing Dimensions.

After these steps, the staging fact columns and dimension values are compared with the current values in the fact table, if any. Adjusting records are created in the fact table so that the fact table now reflects the reported state from the staging table. This is why the term state-like is used. Difference transactions are invented if the numeric fact columns have changed.

If the dimensionality has changed, then the fact is “debooked” and then “rebooked” with the correct dimensionality.

Note: If no fact-column values or dimensionality have changed for a given sskey, then no difference transactions are created and no changes are made to the fact table for that sskey.

If the same `sskey` appears in the staging table with more than one `date_key` , then further adjusting transactions are made in the fact base table as appropriate to bring that table in line with the reported staging rows.

When using this semantic type, it is difficult to ensure that Backlog calculations remain consistent when Orders are not completely closed in the source system. Use the Transactional/State-like/Force Close semantic type if this is the issue.

See "Appendix F, “Components of Built-in Semantics,”" on page 533 for a list of the options that are used in the Statelike semantic type.

Note: State-like facts are not limited to a moment in time, but are relevant until negated. In the absence of negating data, the last state carries forward. However, when a dimension row changes in a slowly changing dimension, the new dimension row is not linked to the existing state unless a new fact row is extracted to force this linkage.

For example, if you populate a fact table with information about which individuals belong to which groups, fact rows are written that link a particular indiv_key to a group_key. If an individual changes, the Slowly Changing Dimension semantic causes a new indiv_key to be created for this individual. This key is now the “latest” for this indiv_key_REAL. However, unless the extraction job also writes a new fact row that links the new version of the indiv_key to the same group, the fact table still references the old data information for the individual in this group. The new individual information is not tied to the group.

Statelike/Error on Stage Duplicates

The Statelike/Error on Stage Duplicates semantic type performs the same actions as the Statelike semantic type when the staging table contains at most one row for any `sskey` . If the staging table contains two rows with the same `sskey` value, then semantic execution fails.

If you know that you are not extracting more than one row for any `sskey` value, then execution of a Statelike/Error on Stage Duplicates semantic can be significantly faster than execution of a Statelike semantic.

See "Appendix F, “Components of Built-in Semantics,”" on page 533 for a list of the options that are used in the Statelike/Error on Stage Duplicates semantic type.

Statelike/Single Delta

The Statelike/Single Delta semantic type performs the same actions as the Statelike semantic type when the staging table contains rows with at most one date for any `sskey` . If, for some `sskey` value, the staging table contains multiple rows with that `sskey` and with different dates, then the semantic processes only the rows with the earliest legal date. Any staging-table row with that `sskey` value and a later date is ignored. If the latest date in the staging table for a given `sskey` is earlier than the latest date for that `sskey` in the base fact table, then no new data is added to the fact table for that `sskey` .

See "Appendix F, “Components of Built-in Semantics,”" on page 533 for a list of the options that are used in the Statelike/Single Delta semantic type.

Streaming Fact

As described in "Streaming Extraction" on page 172, if you are reloading a fact, you can stream data directly into the fact table, bypassing the staging table. The Streaming Fact semantic type creates indexes and performs consistency checks on a fact that are loaded by streaming extraction. You must apply a Streaming Dimension semantic after populating a dimension table with streaming extraction.

See Also, "Appendix F, “Components of Built-in Semantics,”" on page 533 for a list of the options that are used in the Streaming Fact semantic type.

Note: When you use a Streaming Fact semantic instance in an extraction job, the Force rebuild of target table option in the Extraction Steps tab of the Job dialog box is ignored during execution of the semantic. The table is always rebuilt.

Transactional

The Transactional semantic type is the simplest means of moving fact-staging data into fact base tables. The built-in Transactional semantic type uses the following logic:

- For `sskey` s that are already specified in the current fact base table, all rows in the staging table with that `sskey` and with the same or earlier `date_key` are discarded. If the `sskey` already exists in the fact table, only later dates can be added to the fact table.
- Two or more rows with the same `sskey` can be imported into the fact base table if they arrive in the same extraction and the `sskey` either does not already exist, or exists but is dated earlier. (This property is used by the pseudo-order approach to the Booking/Shipping problem; see "Transactional/Statelike/Force Close" on page 251.)
- In the EpiChannel log file, all rows added to the fact base table are reported as the **Inserted** value, and the total number of rows in the staging table is reported as the **Processed** value.

Note the following considerations:

- Transactional semantics must be used for event facts; once the fact event has occurred, it can never be modified.
- To reload transactions after they are loaded into the fact base table, the rows must be deleted from the fact base table, or the fact base table must be truncated. See "Reload Date Fact" on page 248 and "Initial Load Fact" on page 245 for more information.
- The date comparison is always done with respect to `date_key` , which references the built-in date dimension role. Any user-defined dimension roles that refer to the date dimension are ignored.

See "Appendix F, “Components of Built-in Semantics,”" on page 533 for a list of the options that are used in the Transactional semantic type.

Transactional Incremental Streaming

You can insert fact data directly into the INS table, bypassing the staging table, using the Incremental Streaming job type. The Transactional Incremental Streaming semantic type creates indexes and performs minimal consistency checks on a fact that is updated by streaming extraction.

See "Appendix F, “Components of Built-in Semantics,”" on page 533 for a list of the options that are used in the Transactional Incremental Streaming semantic type.

Note: The streaming incremental semantic inserts data directly, without performing any de-duplication checks for the Transactional semantic. Data extracted in this way is written directly to the base tables as-is, which can result in data inconsistencies. This semantic does not prevent this type of data contamination.

Transactional/Dedup Stage

The Transactional/Dedup Stage semantic type removes duplicate values from the staging table before adding transactional data to the fact table.

For every `sskey` that appears in the staging table, rows with dates earlier or the same as the latest date in the base table are ignored. Of the remaining rows (if any), the first one is picked. After duplicates are removed from the staging table in this way, the remaining rows are added as with the Transactional semantic type.

See "Appendix F, “Components of Built-in Semantics,”" on page 533 for a list of the options that are used in the Transactional/Dedup Stage semantic type.

Transactional/Statelike

The Transactional/Statelike semantic type combines the actions of the Transactional and Statelike semantic types. The data in a staging-table row is treated as transactional or state-like, depending on the value in the `process_key` column:

- Rows with a `process_key` value of `1` are treated as transactional and are merged into the fact table in the same way as with the Transactional semantic type.
- Rows with a `process_key` value of `2` are treated as state-like and are merged into the fact table in the same way as with the Statelike semantic type.

When using this semantic type, it is difficult to ensure that Backlog calculations remain consistent when Orders are not completely closed in the source system. Use the Transactional/Statelike/Force Close semantic type instead.

See Also, "Appendix F, “Components of Built-in Semantics,”" on page 533 for a list of the options that are used in the Transactional/Statelike semantic type.

Note: If a table has both transactional and statelike data you must not use the same sskey to refer to both types of facts.

Transactional/Statelike/Force Close

This semantic type is equivalent to Transactional/Statelike with the following additional logic:

Once a Booked Order is specified into the EpiMart database, it remains in that state (with an Open Backlog) forever. In normal scenarios, an invoice eventually arrives, which closes the Backlog. However,

in some source systems, Booked Orders can be removed from the system completely. If such an Order is specified into the EpiMart database, then it remains in an Open Backlog condition forever.

The solution to this problem is to use Transactional/Statelike/Force Close, which establishes a “Challenge Protocol” for Open Orders. In this scenario, all Open Bookings must be extracted into the staging table during every extraction. The reason is that the Force Close logic closes out all Open Orders (`sskey` s with non-zero facts in the system) that do not appear in the fact-staging table. Force Close occurs if the minimum `transtype_key` value is between 1 and 99.

To determine whether an order with a given `sskey` is open, the fact values of all rows with that `sskey` are added. This sum is taken over all values of `transtype_key` . If the sum is not zero, then the order is open. If bookings are recorded with positive fact values and shipments are recorded with negative fact values, then the open orders are exactly those orders for which bookings are not equal to shipments.

When an order is closed by this semantic type, an order entry is constructed with the lowest `transtype_key` value that is used for this `sskey` . Typically, this `transtype_key` is used for bookings. The fact values in this constructed order entry are those values that result in a sum of zero for the fact table rows with the `sskey` of the order being closed.

When using this semantic type, the methodology that is found to work in practice involves the use of pseudo-orders. In this methodology, multiple `sskey` s are used for each order: one `sskey` is used for the original booking and a different `sskey` is used for each shipment. Each shipment `sskey` is used for both the shipments and the associated pseudo-order. For example, you can use the order number for the original booking and the invoice number for each shipment.

After an order is booked, the following extraction steps are used in every extraction, as long as the order is open:

- One extraction command extracts all Open Orders. Fact amounts are the Open amounts (what is not shipped yet), not the ordered amounts. The transaction type for this statement must be in the Booking range (1…99), the `process_key` value must be 2 (state-like), and the `sskey` value must be the same as that of the original order. The date must be the same as the date of the original order.
- The second extraction statement extracts all new shipments. These shipments are recorded with a `process_key` of 1 (Transactional) and with a unique `process_key` value. The fact values of these records must be negative for positive shipments. The transaction type must be in the shipment range (101 or greater), and the date must be the date of shipment.
- The third statement is a restatement of the shipment, but as a Booking (`transtype_key` between 1…99). The `process_key` is still 1 (allowing the Transactional Semantics to import it), but the transaction type is a Booking. The same `sskey` and dimensionality as in the second statement are used. The date must be the same as that of the original booking. These are the pseudo-orders, since they are actual shipments that are specified as Orders.

The net effect of this methodology is as follows:

When a shipment occurs, the open booking quantity (which is extracted in statement 1) is reduced by the number of units shipped. The state-like semantics then creates a difference transaction that reduces the amount of the open order to the new level.

At the same time, a (negative) shipment (with a new `sskey`) is created for the shipped amount by statement 2, and a (positive) pseudo-order in the same amount is created by statement 3. Note that

the convention of recording shipments as negative values ensures that the sum of fact values for the new `sskey` is always zero.

Eventually, when the order is removed from the system, no state-like fact for that `sskey` is generated in statement 1. At this point, if the sum of values for the `sskey` is not zero, then a force-close transaction with values that result in a sum of zero is added to the fact table. Now the fact table has a value of zero for the original `sskey` , and booked and shipped values for the new `sskey` s that correspond to the amount actually shipped.

For example, "Table 40: Order Record In Staging Table" on page 253 illustrates the record for an order that appears in a staging table .

Table 40: Order Record In Staging Table

sskey	Cust	Prod	Date	trans	proc	Qty
123a	23	44	4/11/99	1	1	22

"Table 41: Order Record in Fact Table" on page 253 illustrates how this data is merged into the fact table with transactional logic.

Table 41: Order Record in Fact Table

sskey	Cust	Prod	Date	trans	Qty
123a	23	44	4/11/99	1	22

"Table 42: Extracted Facts for Shipment" on page 253 shows the result of extracting the records for 10 items that is shipped to the customer:

Table 42: Extracted Facts for Shipment

sskey	Cust	Prod	Date	trans	proc	Qty
123a	23	44	4/11/99	1	2	12
234b	23	44	4/14/99	101	1	-10
234b	23	44	4/11/99	1	1	10

The first row is a state-like fact for the part of the order that has not yet shipped. The date is the same as that of the original order. The next two rows are the shipping fact and pseudo-order booking for the part of the order that is shipped. The date for the booking of the pseudo-order is the same as the date for the booking of the original order, and the date for the shipping is the actual date of the shipment.

These facts are then merged into the fact table with transactional and state-like logic, as "Table 43: Shipment Merged into Fact Table" on page 253 illustrates.

Table 43: Shipment Merged into Fact Table

sskey	Cust	Prod	Date	trans	Qty
123a	23	44	4/11/99	1	22

sskey	Cust	Prod	Date	trans	Qty
123a	23	44	4/11/99	1	-10
234b	23	44	4/14/99	101	-10
234b	23	44	4/11/99	1	10

If, in the next extraction, there is no record for `sskey` number 123a, then the force-close logic constructs a record to close the order (see "Table 44: Fact Table with Forced-Closed Order" on page 254). The date of this new record is the same as the latest date for that `sskey` in the fact table. In the pseudo-order approach, this is the same as the date of the original booking.

Table 44: Fact Table with Forced-Closed Order

sskey	Cust	Prod	Date	trans	Qty
123a	23	44	4/11/99	1	22
123a	23	44	4/11/99	1	-10
234b	23	44	4/14/99	101	-10
234b	23	44	4/11/99	1	10
123a	23	44	4/11/99	1	-12

The sum of the **Qty** field for all facts with `sskey 123a` is now zero, and the booking and shipping records are in the fact table with `sskey 234b` . Notice that the bookings and debookings for the original order (`sskey 123a`) all have the same date, so there is no day on which there is a non-zero booking value for this `sskey` . The bookings and shipments for the pseudo-order have the actual booking and shipping dates.

See "Appendix F, “Components of Built-in Semantics,”" on page 533 for a list of the options that are used in the Transactional/Statelike/Force Close semantic type.

Note: If the staging table is not populated, the Force Close option has no effect.

Undo Fact

This semantic is used in a Campaign extraction job template to remove undone campaign information from the fact backfeed P, Q, A, and B tables (see "Campaign Undo" on page 201 for additional information about undoing campaigns).

Fact Table Compression

Fact tables in Infor Campaign Management are normally only changed using INSERT; rows never UPDATE. For this reason, in order to change an existing fact row, a corresponding negating row is first created to cancel the affects of the positive row, and then a new row is created with the updated values.

However, the original row and the negating rows remain in the base tables. (For convenience, the original row and the negating rows are referred to as ‘matter/antimatter’ pairs). These matter/antimatter pairs can consume a lot of disk space even when they don't affect the results of queries. The purpose of this Fact Table Compression is to allow the removal of these rows in a production environment for performance and space-saving reasons.

Accordingly, there is a flag for enabling Fact Table Compression in the Semantic Instance Dialog box. When this option is selected for a fact semantic, the semantic deletes all matter/antimatter rows in the existing base table, in addition to performing the usual operations of the selected semantic (note that this only applies to compiled fact semantics and the new Ignore New Fact semantic).

If the semantic runs in Revise mode, the deletion is performed via DML (a DELETE statement) against the inactive copy of the fact table. In this case, matter/antimatter pairs are only deleted for the previous version of the fact table (that is, new rows from the current or previous extraction are not affected).

If the semantic runs in Rebuild mode (either based on number of changed rows and datamart state, or because the semantic is configured to run in Force Rebuild mode), the table is rebuilt with fact sums, leaving out rows with all-zero sums. In this case, matter/antimatter rows are deleted for the current version of the table, but any new rows from the current extraction are not affected. In addition, if a fact has multiple rows for an ss_key/date/dimensionality combination, but the fact values for those rows have non-zero sums, then the table is rebuilt with a single row for that ss_key/date/dimensionality combination.

In order to facilitate running fact compression without adding new data, there is a new fact semantic called "Ignore New Fact Data" that ignores the contents of the staging table and performs the appropriate rebuild/revise of the base table with no new data (performing fact compression or fusion/fission updates if those options are selected).

Comparison of Built-in Fact Semantic Types

This section illustrates the differences between the default Infor Campaign Management fact semantic types by means of examples from a fictitious data mart containing customer, product, and order data.

After an initial extraction, you can have the data shown in the table below in the fact-staging table.

Table 45: Initial Order Staging Table

ikey	sskey	Cust	Prod	Date	trans	proc	Qty
1	41N	lobt	neat	4/22/99	1	1	4
2	65R	bobb	neat	4/8/99	1	1	6
3	57Q	bobb	wow	4/23/99	1	1	3
4	48Z	lobt	cool	4/7/99	1	1	7
5	91B	lobt	wow	4/1/99	1	1	1
6	46C	betk	cool	4/19/99	1	1	2
7	55T	bobb	cool	4/17/99	1	1	3

Here `Cust` is the Customer `sskey` , `Prod` is the Product `sskey` , `trans` is the transtype key, `Proc` is the process key, and **Qty** is the quantity ordered.

Note: The process key appears only in staging tables for facts that have the Contains Transactional and Statelike Rows option checked in the Fact dialog box.

When this data is loaded into the data mart with an initial load fact semantic, you get a data mart table like "Initial Order Fact Table" on page 256. (Row is not included in actual data mart tables.)

Table 46: Initial Order Fact Table

Row	sskey	Cust	Prod	Date	trans	Qty
1	41N	3	4	4/22/99	1	4
2	65R	2	4	4/8/99	1	6
3	57Q	2	3	4/23/99	1	3
4	48Z	3	2	4/7/99	1	7
5	91B	3	3	4/1/99	1	1
6	46C	4	2	4/19/99	1	2
7	55T	2	2	4/17/99	1	3

Here `Cust` and `Prod` are the integer foreign keys for the dimension table rows that correspond to a fact.

In a subsequent extraction, you can load the following new data into the Order staging table ("New Order Staging Table" on page 256).

Table 47: New Order Staging Table

ikey	sskey	Cust	Prod	Date	trans	proc	Qty
1	57Q	bobb	wow	4/23/99	1	2	3
2	48Z	lobt	cool	4/7/99	1	2	5
3	46C	betk	cool	4/19/99	1	1	1
4	11L	betk	wow	4/7/99	1	1	1
5	65R	bobb	neat	4/8/99	1	2	7
6	71Z	jrd	neat	5/8/99	1	2	5
7	41N	lobt	neat	4/24/99	1	1	3
8	26F	jrd	wow	5/14/99	1	1	10
10	55T	bobb	neat	4/17/99	1	2	3

This data can be merged into the fact tables in different ways, depending on your choice of both fact and dimension semantics.

Transactional

If the new data is merged with a Transactional semantic, then all facts in the staging table are added to the fact table as transactions, unless the fact table already contains a fact with the same `sskey` and the same date or a later date.

The exact structure of the revised fact table depends on the semantic it is used to update the Customer dimension. If the Customer dimension is updated with a First Dimension Value or Latest Dimension Value semantic, then the fact table looks like "Table 48: Order Fact Table, Transactional Revision 1" on page 257 (Row is not included in actual data mart tables).

Table 48: Order Fact Table, Transactional Revision 1

Row	sskey	Cust	Prod	Date	trans	Qty
1	41N	3	4	4/22/99	1	4
2	65R	2	4	4/8/99	1	6
3	57Q	2	3	4/23/99	1	3
4	48Z	3	2	4/7/99	1	7
5	91B	3	3	4/1/99	1	1
6	46C	4	2	4/19/99	1	2
7	55T	2	2	4/17/99	1	3
8	11L	4	3	4/7/99	1	1
9	71Z	5	4	5/8/99	1	5
10	41N	3	4	4/24/99	1	3
11	26F	5	3	5/14/99	1	10

Note the following considerations:

- The value of `process_key` is ignored. All facts are assumed to be transactional.
- If a staging table row has the same `sskey` as a fact table row and the same date value as or an earlier date value than some row of the fact table, then that staging table row is not added to the fact table. For example, row 3 of the staging table is not added to the fact table because it has the same `sskey` and the same date as row 6 of the fact table.
- All other staging table rows are added to the fact table.

If the Customer dimension is updated with a Slowly Changing Dimensions semantic, then the fact table looks like "Table 49: Order Fact Table, Transactional Revision 2" on page 257.

Table 49: Order Fact Table, Transactional Revision 2

Row	sskey	Cust	Prod	Date	trans	Qty
1	41N	3	4	4/22/99	1	4
2	65R	2	4	4/8/99	1	6

Row	sskey	Cust	Prod	Date	trans	Qty
...	...	...	...	...	...	...
9	71Z	5	4	5/8/99	1	5
10	41N	5	4	4/24/99	1	3
11	26F	6	3	5/14/99	1	10

The important difference here is that the new order for Lobby Tableman (row 7 of the staging table, row 10 of the fact table) is specified with a customer key value that points to the dimension table row with updated values (row 5 of the Customer table).

The existing orders for Lobby Tableman still have customer keys that point to the dimension table row with old values (row 3 of the Customer table).

Transactional/State-Like

If the new data is merged with a Transactional/State-Like semantic, then transactional facts are added as they are with a Transactional semantic type. State-like facts (that is, facts with a `process_key` value of `2`) are assumed to reflect the current state of the fact quantities for a given `sskey`. If the fact table values do not match those in the state-like fact, then transactions are added to the fact table to adjust those values.

The structure of the revised fact table again depends on the semantic that is used to update the Customer dimension.

If the Customer dimension is updated with a First Dimension Value or Latest Dimension Value semantic, then the fact table looks like "Table 50: Order Fact Table, Transactional/State-Like Revision 1" on page 258.

Table 50: Order Fact Table, Transactional/State-Like Revision 1

Row	sskey	Cust	Prod	Date	trans	Qty
1	41N	3	4	4/22/99	1	4
2	65R	2	4	4/8/99	1	6
3	57Q	2	3	4/23/99	1	3
4	48Z	3	2	4/7/99	1	7
5	91B	3	3	4/1/99	1	1
6	46C	4	2	4/19/99	1	2
7	55T	2	2	4/17/99	1	3
8	48Z	3	2	4/7/99	1	-2
9	11L	4	3	4/7/99	1	1
10	65R	2	4	4/8/99	1	1

Row	sskey	Cust	Prod	Date	trans	Qty
11	71Z	5	4	5/8/99	1	5
12	41N	3	4	4/24/99	1	3
13	26F	5	3	5/14/99	1	10
14	55T	2	2	4/17/99	1	-3
15	55T	2	4	4/17/99	1	3

Note the following considerations:

- Staging table rows with a `process_key` value of `1` are added to the fact table as with the Transactional semantic type.
- If a staging table row with a `process_key` value of `2` has the same dimensionality (that is, the same dimension key values) and the same `sskey` value as an existing fact table row, but a different fact value, then a difference row is added to the fact table. For example, row 5 of the staging table has the same `sskey` and dimensionality values as row 2 of the fact table, but the order quantity is 7 rather than 6. Row 10 of the revised fact table therefore has an order for a single unit, so that the orders for `sskey 65R` add up to 7. Similarly, row 8 updates the value in row 4.
- If a staging table row with a `process_key` value of `2` has the same `sskey` value as an existing fact table row but different dimensionality, then the old order is debooked and the new order is booked. For example, row 10 of the staging table has the same `sskey` as row 7 of the fact table, but a different product dimension key. Therefore, row 14 of the revised fact table debooks the order with the old dimension values, and row 15 rebooks it with the new dimension values.
- If a staging table row with a `process_key` value of `2` has the same dimensionality, the same `sskey` value, and the same fact values as an existing fact table row, then that staging table row is ignored. For example, row 1 of the staging table duplicates row 3 of the fact table, so it is ignored.

If the Customer dimension is updated with a Slowly Changing Dimensions semantic, then the fact table looks like "Table 51: Order Fact Table, Transactional/State-Like Revision 2" on page 259.

Table 51: Order Fact Table, Transactional/State-Like Revision 2

Row	sskey	Cust	Prod	Date	trans	Qty
1	41N	3	4	4/22/99	1	4
2	65R	2	4	4/8/99	1	6
3	57Q	2	3	4/23/99	1	3
4	48Z	3	2	4/7/99	1	7
5	91B	3	3	4/1/99	1	1
6	46C	4	2	4/19/99	1	2
7	55T	2	2	4/17/99	1	3
8	48Z	3	2	4/7/99	1	-7
9	48Z	5	2	4/7/99	1	5

Row	sskey	Cust	Prod	Date	trans	Qty
10	11L	4	3	4/7/99	1	1
...	...	...	...	...	...	...
15	55T	2	2	4/17/99	1	-3
16	55T	2	4	4/17/99	1	3

The difference here is in rows 8 and 9. Row 2 of the staging table refers to the same customer as row 4 of the fact table. However, since the data for Lobby Tableman has changed and the dimension are updated with a Slowly Changing Dimensions semantic, a new row with revised data are added to the dimension table. Since the state-like fact refers to a different dimension table row than that referred to by the initial transactional fact, it is treated as a state-like fact with changed dimensionality. Therefore, the original order is debooked and the new order is rebooked.

Transactional/State-Like/Force Close

If the new data is merged with a Transactional/State-Like/Force Close semantic, then new data is merged in the same way as with Transactional/State-Like. In addition, the Force Close logic is applied to facts with `sskey` s that do not appear in the staging table. When a fact table `sskey` does not appear in the staging table, and there is at least one row in the fact table with that `sskey` and an Order transaction type (that is, with a `transtype_key` value in the range `1 - 99`), then that `sskey` is forced closed. For such an `sskey` , all facts with the `sskey` (regardless of transtype) are added. If this sum is not zero, then a new fact that forces the sum to be zero is added to the fact table. Note that if all fact table rows with a given `sskey` value have `transtype_key` values greater than `99` , then that fact is not affected by the force close logic.

The Transactional/State-Like/Force Close semantic type is most commonly used with the pseudo-order approach described in "Transactional/Statelike/Force Close" on page 251, but it can be applied to any fact and staging tables. When applied to the example fact and staging tables in this section, the exact structure of the revised fact table again depends on the semantic it is using to update the Customer dimension.

If the Customer dimension is updated with a First Dimension Value or Latest Dimension Value semantic, then the fact table looks like "Table 52: Order Fact Table, Transactional/State-Like/Force Close Revision 1" on page 260.

Table 52: Order Fact Table, Transactional/State-Like/Force Close Revision 1

Row	sskey	Cust	Prod	Date	trans	Qty
1	41N	3	4	4/22/99	1	4
...	...	...	...	...	...	...
5	91B	3	3	4/1/99	1	1
6	46C	4	2	4/19/99	1	2
7	55T	2	2	4/17/99	1	3

Row	sskey	Cust	Prod	Date	trans	Qty
...	...	...	...	...	...	...
14	55T	2	2	4/17/99	1	-3
15	55T	2	4	4/17/99	1	3
16	91B	3	4	4/1/99	1	-1

Note the following considerations:

- Since there is no row in the staging table with `sskey 91B`, row 16 of the revised fact table de-books the order in row 5.
- The order in row 6 is not de-booked, because row 3 of the staging table has the same `sskey`.
- All staging table rows are treated as they are with a Transactional/State-Like semantic type. In particular, row 3 of the staging table is not added to the fact table, because its date is before that of row 6 of the fact table.

If the `Customer` dimension is updated with a Slowly Changing Dimensions semantic, then the fact table looks like "Table 53: Order Fact Table, Transactional/State-Like/Force Close Revision 2" on page 261.

Table 53: Order Fact Table, Transactional/State-Like/Force Close Revision 2

Row	sskey	Cust	Prod	Date	trans	Qty
1	41N	3	4	4/22/99	1	4
...	...	...	...	...	...	...
7	55T	2	2	4/17/99	1	3
8	48Z	3	2	4/7/99	1	-7
9	48Z	5	2	4/7/99	1	5
10	11L	4	3	4/7/99	1	1
...	...	...	...	...	...	...
15	55T	2	2	4/17/99	1	-3
16	55T	2	4	4/17/99	1	3
17	91B	3	3	4/1/99	1	-1

The only difference here is that rows 8 and 9 de-book and re-book the order with `sskey 48Z`, as with the Transactional/State-Like semantic type.

Reload Date Fact

If the new data is merged with a Reload Date Fact semantic, then all facts in the staging table are placed into a new fact table. In addition, all facts from the old fact table that have a date that is earlier than the earliest date in the new fact table are appended to the new fact table.

The exact structure of the revised fact table again depends on the semantic it is using to update the Customer dimension.

If the Customer dimension is updated with a First Dimension Value or Latest Dimension Value semantic, then the fact table looks like "Table 54: Order Fact Table, Reload Date Fact Revision 1" on page 262.

Table 54: Order Fact Table, Reload Date Fact Revision 1

Row	sskey	Cust	Prod	Date	trans	Qty
1	57Q	2	3	4/23/99	1	3
2	48Z	3	2	4/7/99	1	5
3	46C	4	2	4/19/99	1	1
4	11L	4	3	4/7/99	1	1
5	65R	2	4	4/8/99	1	7
6	71Z	5	4	5/8/99	1	5
7	41N	3	4	4/24/99	1	3
8	26F	5	3	5/14/99	1	10
9	55T	2	4	4/17/99	1	3
10	91B	3	3	4/1/99	1	1

This is essentially the staging table with older fact-table data appended.

If the Customer dimension is updated with a Slowly Changing Dimensions semantic, then the fact table looks like "Table 55: Order Fact Table, Reload Date Fact Revision 2" on page 262.

Table 55: Order Fact Table, Reload Date Fact Revision 2

Row	sskey	Cust	Prod	Date	trans	Qty
1	57Q	2	3	4/23/99	1	3
2	48Z	5	2	4/7/99	1	5
3	46C	4	2	4/19/99	1	1
4	11L	4	3	4/7/99	1	1
5	65R	2	4	4/8/99	1	7
6	71Z	6	4	5/8/99	1	5
7	41N	5	4	4/24/99	1	3

Row	sskey	Cust	Prod	Date	trans	Qty
8	26F	6	3	5/14/99	1	10
9	55T	2	4	4/17/99	1	3
10	91B	3	3	4/1/99	1	1

The only difference here is in the dimension keys for the new values. Notice that the new entries for Lobby Tableman have key `5`, while the entry (in row 10) is imported from the old fact table still has a key value of `3`.

Backfeed Semantic Types

Backfeed semantics merge recent campaign backfeed data into the main backfeed tables of the EpiOp database. The built-in Backfeed and BackfeedRolloff extraction jobs allow you to perform this merge automatically. See "The Backfeed Job" on page 321 and "The Backfeed Roll-Off Job" on page 324 for additional information.

Fusion and Fission

In some cases, you can have several source-system records that you wish to treat as the same dimension element. For example, you can have used a third-party data-cleansing tool to identify multiple records that refer to the same customer. In such a case, you can fuse these records into a single dimension element.

A fusion/fission semantic combines multiple records into a single representative record (fusion). When you choose to remove a group of fused records, the records are separated into individual records and are once again treated as distinct dimension elements (fission).

Note: Any custom semantics with a Record Fusion or Record Fusion/Fission value for the Rewrite Rows option are deprecated in Infor Campaign Management version 6.5 and later. Such semantics will continue to work as before, but Infor recommends using the Latest Dimension Value With Fusion/Fission semantic to perform fusion and fission, as described in this section.

The Fusion Tables

To fuse dimension records, you must populate a fusion staging table for that dimension (see "Base Dimension Staging SQL Statements" on page 285). This table identifies records that are to be fused and, for each group of fused records, specifies the dimension element that represents that group (the parent element).

Fusion semantics merge information in the fusion staging table into a fusion mapping table . The fusion mapping table contains data about all currently fused records. To remove a group of fused records from the fusion mapping table you can extract the group into a fusion staging table with a `fusion_flag` value of `0` . The fusion semantic fissions the parent-child pairs that appear in the fusion staging table with a `fusion_flag` value of 0. Note that fissioned elements are given the values that they have before fusion is performed. Changes that occurred after these elements are fused are not automatically incorporated.

Note the following fusion table requirements:

- A parent element in a fusion staging table cannot also be a child in the same table. However, a record that is a parent in the fusion mapping table can be a child in the fusion staging table. If this happens, all children of the old parent become children of the new parent.
- A child element cannot have more than one parent. An exception is generated if the fusion table contains records for a child with two parents, and a unique index violation is generated if a record in the fusion staging table conflicts with an element that is already in the fusion mapping table.

Configuring Fusion Extraction Jobs

Fusion requires fact table updates or a complete rebuild of the fact table, depending on the specified revision percentage of the fact table and the number of records to be fused or fissioned. Both updates and rebuilds can be significantly time-consuming, so you can choose to leave out fusion during an extraction:

- If you leave out fusion in an extraction job, use the **Latest Dimension Value, Preserve Fusion** semantic for all fused dimension tables.
- Before running a fusion-preserving job, you must ensure that, for any fact that joins to a fused dimension, the A and B tables have both been properly fused (see "Data Mart Mirroring: A and B Tables" on page 191 for a discussion of A and B tables). You can do this with a job that runs fact fusion semantics on all such facts, commits (for example, using the Rolloff Checkpoint extraction group), and then again runs fact fusion semantics on all such facts. Alternatively, you can run the fact semantic in Force Rebuild mode in the first non-fusion extraction job.
- If you run a fusion extraction job, use the **Latest Dimension Value With Fusion/Fission** semantic for all fused dimension tables in the regular dimension extraction process. Run a fact fusion extraction step after the normal dimension extraction process to update real keys in the fact table. (Normally the fact fusion semantic is run as part of the normal fact extraction process after dimension extraction).
- In the fact fusion extraction step, apply a fact fusion semantic instance to all facts that have at least one list-producing fusion-enabled dimension in their dimensionality. You can enable fact fusion for a semantic instance by setting the value of the **Fact Fusion/Fission** option in the Semantic Instance dialog box to 1. Do not perform fact fusion if you are fusing only non-list producing dimensions in the dimensionality of the fact.

Note: The Fact Fusion/Fission option cannot be used with the Reload Date Fact or First/Last Fact semantic types.

Note: When using the Fact Fusion/Fission option with an Initial Load or Streaming semantic type, you can receive a job validation warning indicating that the semantic has an option set that do not affect semantic behavior. This warning can safely be ignored.

- Run AggBuilder, a MomBuilder extraction step with the **Force Rebuild** option set to 1, and the End of Extraction group as the final steps of a fusion job.

  Note: Do not use any dimension semantics except Latest Dimension Value with Fusion/Fission, Latest Dimension Value, Preserve Fusion, and Initial Load semantics on a fused dimension.

Limitations to Fusion/Fission

Fusion/fission semantics are subject to the following limitations:

- In a fusion staging table, a parent element cannot itself be a child of another parent record in the same table.
- Dimensions to which fusion is applied must be versioned (in the Dimension dialog box, **Options** tab, the **This dimension is non-versioned** option must be unselected).
- If your fact table is partitioned, rebuild/revise decisions are specified for an entire fact table, rather than for each fact table partition.
- When fission is performed, fissioned elements are given the values that they had before fusion was performed. Changes that occurred after these elements are fused and not incorporated automatically.
- If a fused dimension is associated with another dimension via a many--to-one relation, then any two fused elements of the fused dimension must be related to the same element of the related dimension. If two dimension elements that are being fused are not related to the same dimension element, then the fact underlying the relation must be updated when fusion is performed. To update the relation's underlying fact, extract new fact data that creates the appropriate associations and use a Statelike semantic to update the fact.

Parallelism in Semantic Extraction Jobs

Extraction jobs in some customer sites can take a very long time to complete. This delay is largely due to the fact that all of the semantic steps, by default, are executed sequentially.

In order to improve performance through concurrency in semantics, semantic job steps in a Semantic Group are also allowed to be concurrently executed. Consequently, a flag associated with Semantic Groups appears on the UI allowing you to enable parallelism. Set this flag only if all extraction steps in the group are semantic instances, and either all semantic instances in the group are of type `dimension` or all semantic instances in the group are of type `fact`. Infor Campaign Management does not support mixing of type `dimension` or type `fact` in parallel semantic execution mode.

A Parallel Semantic Group can include any number of semantics, but the maximum number of semantics that are executed at any one time is determined by the value of the NumThreads configuration key, found in the configuration settings under Optimization/EpiChannel. To enable semantic parallelism, ensure that this key is set to a value greater than 1.

Figure 8-1: Parallel Semantic Group

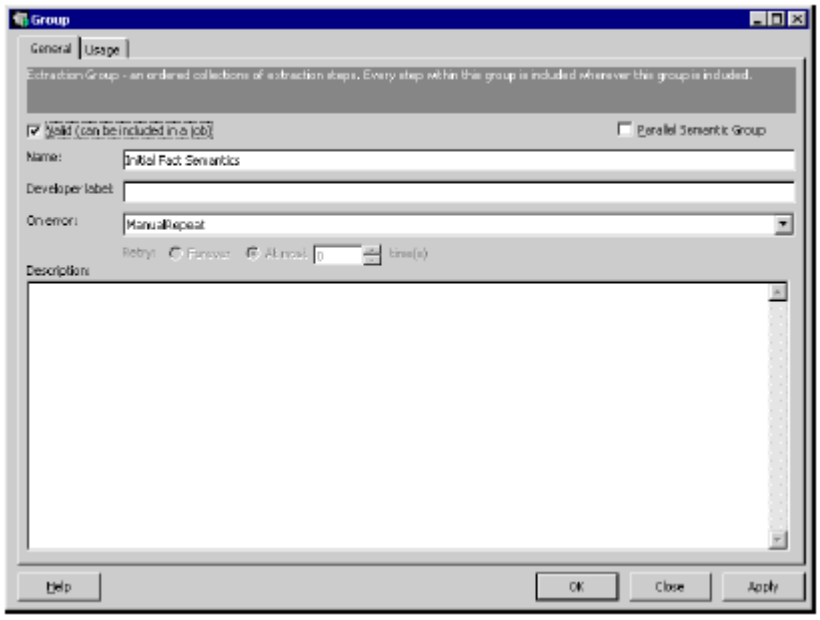


Figure 83: Parallel Semantic Group

By constraining the parallelism to a Semantic Group, the execution time of the group is therefore bound to the longest step, which facilitates tracking, recording, and handling the execution progress.

Note:

A Parallel Semantic Group:

- Contains only semantic instance job steps
- Must contain either only dimension semantic job steps or only fact semantic job steps (not both)
- Does not contain two or more semantics that operate on the same table.

Defining New Semantic Types

If you have special business logic that is not reflected in any of the pre-defined semantic types, you can use Admin Manager to define new semantic types for your EpiCenter.

1 Right-click the **Semantic Types** subfolder of the **Extraction** folder for your EpiCenter. Select **New Semantic Type** to display the Semantic Type dialog box.

Figure 8-1: Semantic Type Dialog Box

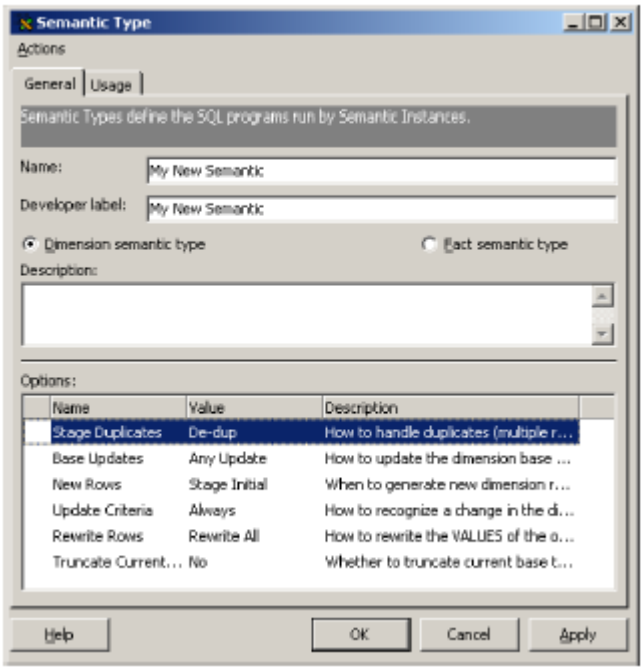


Figure 84: Semantic Type Dialog Box

2 Enter a name and description for the new semantic type.

3 Select **Dimension** or **Fact** semantic type.

4 Select values for all of the options in the grid control. The available options are described in "Components of Dimension Semantics" on page 225 and "Components of Fact Semantics" on page 239.

5 Click **OK** to save the new semantic type. When you save the semantic type, you are also given the option of saving a copy of the new semantic template to a file. Saving a copy of the template to a file in addition to importing it into your EpiCenter allows you to import the template into another EpiCenter at a later time.

Extraction Steps

9

The actual work of an extraction job is done in global extraction steps, which include steps for SQL extraction code and for semantic instances. A single global extraction step can be used in several different jobs.

By default, every EpiCenter includes a default set of extraction steps that you can use when configuring your extraction jobs (see "Extraction Job Templates" on page 319). In addition to the default extraction steps, you need to define additional extraction steps based on the structure of your source systems and your EpiCenter.

Note: Do not copy and paste scripts, commands or code from this document. Line breaks and some other characters picked up when you copy from the PDF can cause errors in the pasted text.

The Extraction Steps Dialog Box

You use the Extraction Steps dialog box (see "Figure 85: Extraction Steps Dialog Box" on page 270) to configure and group global extraction steps. In this dialog box, all extraction steps are grouped into a top-level extraction group, which is called All Extraction Steps .

To open the Extraction Steps dialog box, double-click the **Edit Extraction Steps** icon in the Extraction folder of your EpiCenter. You can also open the Extraction Steps dialog box by right-clicking the Extraction folder of your EpiCenter and selecting **Edit Extraction Steps** from the pop-up menu, or by clicking on the Edit Extraction Steps tool-bar icon.

To create a new extraction step select the extraction group to which you like to add the step and click **New** . When you click **New**, a menu with available extraction-step types drops down. You can select the type of extraction step that you like to create.

Figure 9-1: Extraction Steps Dialog Box

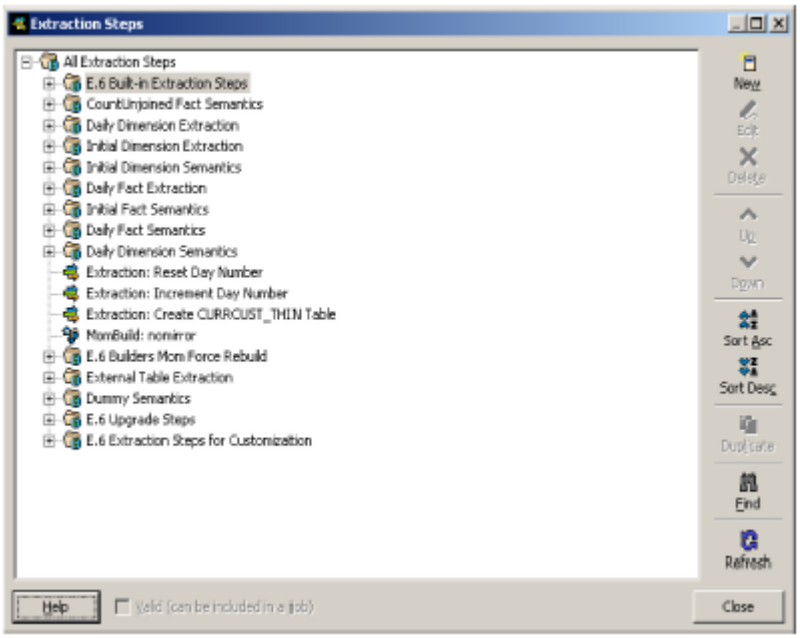


Figure 85: Extraction Steps Dialog Box

After you have created an extraction step, you can select it in the Extraction Steps dialog box and click **Edit** to modify it, or click **Remove** to delete it. Keep in mind that any changes that you specify affects all jobs that use this extraction step.

To disable an extraction step in all extraction jobs that use it, select the extraction step and unselect the **Valid (can be included in a job)** option. The disabled extraction step is greyed out with a yellow exclamation point.

To specify a copy of an existing extraction step, select the step and click **Duplicate** . Clicking **Duplicate** opens a dialog box in which you can enter a new name for the duplicate extraction step.

Managing Extraction Steps

Over time, EpiCenters tend to collect extraction steps that are not used. These extra steps are either the result of experimentation, changes in source systems, or upgrade.

Some unused extraction steps can produce Scrutiny errors or prevent the execution of EpiChannel or the Infor Campaign Management Server. You can disable these extraction steps in the Extraction Steps dialog box, or in the Extraction Steps pane of the Job dialog box.

EpiMeta upgrade automatically disables all upgraded extraction steps. You can quickly enable these steps and create a list of candidate steps for deletion by using the following procedure.

Enabling Upgraded Extraction Steps

1. Disable any job that you do not use in your EpiCenter. To disable a job, open the Job dialog box for the job and unselect the **Enabled (Can be executed)** option.
2. In the **Actions** menu of the Job dialog box, select **Mark Global Steps Valid or Invalid > Mark All Global Steps Invalid** . This action disables all extraction steps in your EpiCenter.
3. In the **Actions** menu of the Job dialog box, select **Mark Global Steps Valid or Invalid > Mark All Global Steps Used in this Job Valid** to enable the extraction steps that are used in the current job.

   Alternatively you can select **Mark Global Steps Valid or Invalid > Mark All Global Steps Used All Enabled Jobs Valid** to enable the extraction steps that are used in the jobs that you have enabled.
4. In the **Actions** menu of the Job dialog box, select **Mark Global Steps Valid or Invalid > List Invalid or Unused Global Steps** to create a list of all extraction steps that are not included in your enabled extraction jobs. Consider deleting these steps, especially those that are built into prior Infor Omni-Channel Campaign Management releases and which you did not alter for your installation.

Extraction Step Types

The following basic extraction step types are available:

- Extraction Command
- Group
- Semantic Instance
- System Call

You can also define any of the following extraction utility steps:

- AggBuild
- Cancel
- Cancel Backfeed
- Commit
- Commit Backfeed
- Create Current Views
- Dimension Backfeed Semantics
- Fact Backfeed Semantics
- Fact Compression
- MomBuild
- Refresh
- Rolloff
- Rolloff Backfeed Semantics
- Truncation

The remainder of this chapter describes how to define and configure each of these extraction step types.

Extraction Commands

Extraction commands do the work of copying data from source systems and moving data into external tables and staging tables. Admin Manager provides a graphical tool for defining extraction commands.

You can also use a SQL statement to define an extraction command. Admin Manager provides SQL code templates for staging tables and external tables. These templates must be customized based on the structure of your source systems.

Note: Do not use an ODBC data store for SQL execution job steps. Instead, configure the job steps to use SQL Server, Oracle, or DB2 data stores instead.

ODBC data stores are only supported as extraction source databases. They can be used in the extraction commands that read in data from ODBC data sources and populate facts, dimensions, and external tables with this data. However, they are not supported for use as data stores for SQL execution job steps that do not populate these tables. Doing so results in a "no suitable driver" error.

Extraction commands are executed against the input data store that is assigned in the Job dialog box, and the results are stored in the output data store.

You can use the Extraction Wizard to define a new extraction command. To define an extraction command using the wizard, take the following steps.

Defining a New Extraction Command Using the Extraction Wizard

1. In the Extraction Steps dialog box, select the group to which you wish to add your new extraction command and create a new Extraction Command step, as described on page "The Extraction Steps Dialog Box" on page 269.
2. In the wizard, select a type of table to populate.
3. Select the table that you wish to populate.
4. Select the data store from which you wish to populate this table.
5. Select the source-system tables that you wish to use to populate the selected EpiCenter table. All external tables in the selected data store are available.
6. Enter a name and developer label for the extraction command.
7. Click **Finish** to define the extraction command. Check **Open this Extraction Command in the Extraction Command Editor Dialog** to go immediately to the Extraction Command dialog box to edit the extraction command that you have just defined.

   Note the following:

   - If you do not wish to use the wizard, click **Create Extraction Command with Editor** in the first screen of the wizard to open the Extraction Command dialog box for a new extraction command.

- If you wish to disable the wizard, check **By Default Don't Start With This Wizard** in the wizard dialog box and then close the dialog box. After you have done this, all new extraction commands are immediately opened in the Extraction Command dialog box.

Configuring an Extraction Command

You configure an extraction command in the Extraction Command dialog box. If you have defined an extraction command with the wizard, some options are already configured for you, but you can change any of these options if necessary.

1 Double-click the extraction command to open the Extraction Command dialog box (see "Figure 86: Extraction Command Dialog Box: General Tab" on page 273).

Figure 9-2: Extraction Command Dialog Box: General Tab

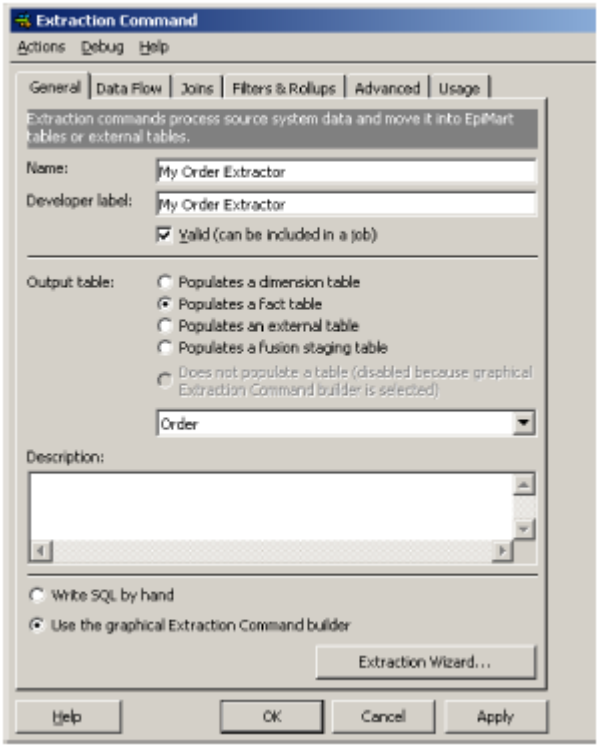


Figure 86: Extraction Command Dialog Box: General Tab

- If you wish to use the wizard to define the extraction command, click **Extraction Wizard**.
- If you wish to use the wizard whenever you define a new extraction command, ensure that the **By Default, Don't Start With This Wizard** check box is not checked on the first screen of the wizard.

2 In the **General** tab, enter a **Name** and **Developer label** for the extraction command.

3 Select an option for the results of the extraction command:

- Select **Populates a Dimension Table** to place the results into a dimension staging table.
- Select **Populates a Fact Table** to place the results into a fact staging table.
- Select **Populates an External Table** to place the results into an external table.
- Select **Populates a Fusion Staging Table** to place the results into a fusion staging table.
- Select **Does Not Populate a Table** to discard the results. Select this option for SQL code that is used only for its side effects. By default, the statement is executed against the output data store. Select **Execute Against Input Data Source** in the **Advanced** tab to execute the statement against the input data store.

Note: To refer to a fact, dimension, or external table without populating it, select the Populates option for the type of table to which you wish to refer and check Run statement, but do not save results to a table in the Advanced tab.

4 If the step references a table, select the table from the drop-down list box.

If the extraction step populates a table, then this is the fact or dimension table for which a staging table is to be populated, or the external table that is to be populated.

5 Select **Use the Graphical Extraction Command Builder** to define this extraction command with the graphical extraction-command builder (see "Using the Graphical Extraction-Command Builder" on page 277). Select **Write SQL by Hand** to define this extraction command with an SQL statement (see "Using SQL to Define Extraction Commands" on page 284).

Note: If you choose to define an extraction command using an SQL statement, you can not edit the command with the graphical extraction-command builder.

If you choose to define an extraction command with the graphical extraction-command builder, you can later edit the SQL code for that extraction command. However, you can not edit the command with the graphical extraction command builder again after editing the SQL code.

6 In the **Advanced** tab, select the action to be taken On Error. (See "Table 60: Possible Actions On Error" on page 301.)

Figure 9-3: Extraction Command Dialog Box: Advanced Tab

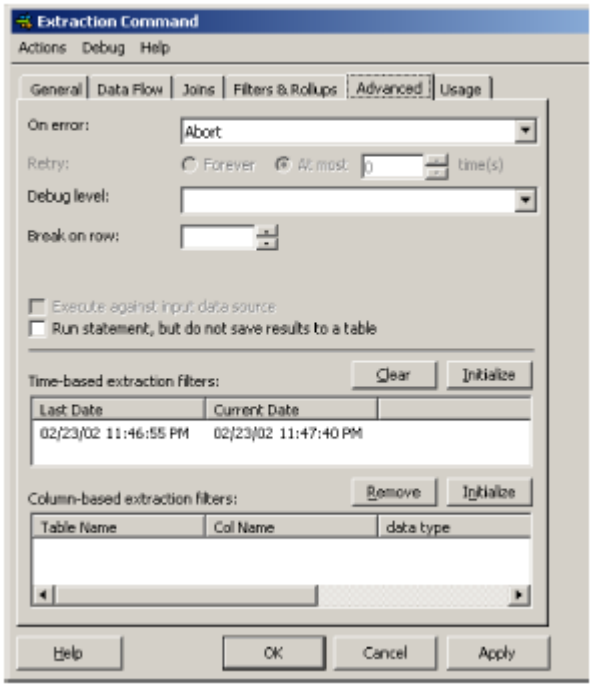


Figure 87: Extraction Command Dialog Box: Advanced Tab

Note: The Infor Campaign Management Server issues queries against external tables when using external lookup filters. (See “Attributes” in chapter 5 of the *Infor Campaign Management Topic Implementation Guide* for a discussion of external lookup filters.)

7 If your EpiMart database resides on a DB2 data store, select **LoadWriter** as the **Bulk Load Option** for the extraction command. See "Bulk Loading in DB2" on page 197, for more information on these DB2 bulk loading options.

8 Select **Run statement, but do not save results** to reference an EpiMart table without populating the corresponding staging table or external table.

This option is only available when you have selected one of the **Populates** options in the **General** tab. Select **Execute Only** when SQL is to be expanded to reflect the structure of one of these tables, but the returned rows from the SQL statement must be discarded.

9 Select **Execute Against Input Data Source** to execute an SQL statement against the input data store rather than the output data store. This option is only available if **Does Not Populate a Table** is selected in the **General** tab.

10 If you select **Interactive** from the **Debug Level** drop-down list box, then execution enters interactive mode after the row number specified in **Break on Row** . A **Break on Row** value of 0 indicates that the entire extraction command must run in interactive mode. See "Setting Breakpoints" on page 373.

Extraction Filters

The Infor Campaign Management extraction filters allow you to extract only those rows that satisfy a specified criterion, such as having a date or column value in a certain range. For example, you can apply an extraction filter that extracts only those source-system rows that have changed since the last extraction.

When you run an extraction step that contains an extraction filter, specified source-system values are compared to values that are recorded when the extraction step is last executed. Only those values that match the filter criterion are actually extracted from the source system. At the end of the extraction step, new values are recorded.

You apply extraction filters using the Infor Campaign Management extraction-set identification macros in the **SQL Statement** tab of the Extraction Command dialog box. See "Extraction Macros" on page 480 for more information on these macros.

Initializing Extraction Filters

If an extraction command can use extraction filters, you must initialize those filters before using the extraction command. Initializing extraction filters ensures that data is properly filtered the first time that the extraction command is used.

1. To initialize time-based extraction filters that are used in the **SQL Statement** tab of the Extraction Command dialog box, click **Initialize** in the Time-Based Extraction Filters pane of the **Advanced** tab.
2. To initialize column-based extraction filters that are used in the **SQL Statement** tab of the Extraction Command dialog box, click **Initialize** in the Column-Based Extraction Filters pane of the **Advanced** tab of the Extraction Command dialog box.

   When you initialize extraction filters, entries for those filters are displayed in the appropriate pane of the **Advanced** tab of the Extraction Command dialog box.

Viewing Extraction Filter Status

The status of extraction filters that are applied to extract can be seen in the lower half of the **Advanced** tab of the Extraction Command dialog box (see "Figure 87: Extraction Command Dialog Box: Advanced Tab" on page 275):

1. The upper pane shows the values that are recorded for **Time Based** extraction filters for this extraction step.
2. The lower pane shows the values that are recorded for **Column Based** extraction filters.

   To clear time-based extraction-filter values, select the extraction filter and click the **Clear** button in the Time-Based Extraction Filters pane of the **Advanced** tab of the Extraction Command dialog box. You must reinitialize time-based extraction filters after clearing these values.

   To remove a column-based extraction-filter, select the filter that you wish to remove in the Column-Based Extraction Filters pane of the **Advanced** tab of the Extraction Command dialog box and click the **Remove** button. If the filter that you have removed is still used in your extraction code, then you must reinitialize column-based extraction filters after removing the filter.

Viewing and Checking SQL

The **Debug** menu in the Extraction Command dialog box allows you to view and verify generated SQL code.

Select **Check SQL** from the **Debug** menu to check the syntax of the SQL code that is issued by an extraction command. If you are using SQL code to define your extraction command, then your SQL code is checked for syntactic correctness. If you are using the graphical extraction-command builder, then the SQL code that is generated by the extraction command is checked for syntactic correctness.

Note: In order to ensure correctness, SQL syntax is verified using the database engine of the source system that you have specified in the General tab. Therefore, you must be connected to the source data store in order to check the syntax of your extraction command.

If you are using the graphical extraction-command builder, you can select **View SQL** from the **Debug** menu to view the SQL code that is issued against the source data store by an extraction command. If you are using SQL code to define an extraction command, the **View SQL** menu item is disabled.

Select **View Output** from the **Debug** menu to issue a query against the source database and display sample rows that are returned by the extraction command.

Select **SQL Query Window** from the **Debug** menu to open a query window from which you can issue SQL queries against the source database.

Date Fields

Infor Campaign Management uses the following standard input format for dates:

mm/dd/yyyy

Replace mm with the two-digit month. Replace dd with the two-digit day. Replace yyyy with the four-digit year.

Note: The above format for dates applies to all languages and locales.

Using the Graphical Extraction-Command Builder

To use the Infor Campaign Management graphical extraction-command builder, select **Use the Graphical Extraction Command Builder** in the **General** tab of the Extraction Command dialog box. When you select this option, the Extraction Command dialog box has tabs for **Data Flow** , **Joins** , and **Filters & Rollups** , in addition to the **General** , **Advanced** and **Usage** tabs.

If you have used the extraction-command wizard to define the extraction command, then some of the data flow and joins are already defined based on column names and foreign keys in the selected source-system tables.

Data Flow

The **Data Flow** tab of the Extraction Command dialog box (see "Figure 88: Extraction Command Dialog Box: Data Flow Tab" on page 278) allows you to configure how data is copied from source-system tables

to the destination table. Arrows show how data is copied from source-table columns to destination columns.

As shown in Figure "Figure 88: Extraction Command Dialog Box: Data Flow Tab" on page 278, the **Data Flow** tab displays data flow as follows:

- If data from a single source-table column is extracted to a destination column, then an arrow is shown from the source to the destination.
- If data from more than one source-table column is extracted to a destination column, then arrows are shown from all source columns to an intermediate node, and another arrow is shown from the intermediate node to the destination.

Figure 9-4: Extraction Command Dialog Box: Data Flow Tab

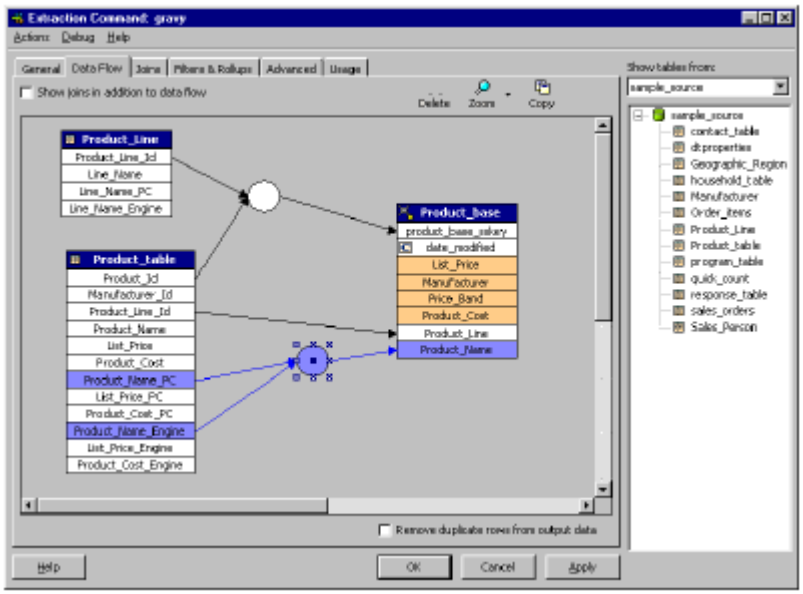


Figure 88: Extraction Command Dialog Box: Data Flow Tab

- If a destination column is assigned a value that does not depend on the values of source-table columns, such as a constant or the current date, then the letter C is displayed in that column.
- If a destination column has not yet been assigned a value, then that column is displayed with an orange background.

If you select a destination column or an intermediate node, then the destination column and all source columns from which data is being extracted are highlighted with a blue background, as shown in "Figure 88: Extraction Command Dialog Box: Data Flow Tab" on page 278. If you select a source column, then the source column and all destination columns to which data is being extracted are highlighted with a blue background.

Defining New Data Flow

1 Select the name of a source-table column. The column is highlighted and a dot appears in the middle (see "Figure 89: Selected Source Column" on page 279).

2 Click on the dot in the middle of the column and drag to the desired column of the destination table. When you do this, an arrow appears from the source to the destination.

   If an arrow from another column is already going to the destination column, then an intermediate node is automatically created. By default, multiple source-column values are concatenated.

3 Double-click the intermediate node to edit the function that is applied to the source data.

4 To apply a function to a source data value, or to define a destination-column value that does not depend on your source data (such as a constant or the current date), double-click the destination column. This action opens the SQL expression builder (see "Using the Graphical Extraction-Command Builder" on page 277), in which you can define the appropriate function.

Figure 9-5: Selected Source Column

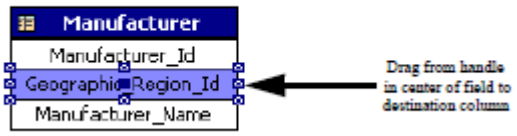


Figure 89: Selected Source Column

5 To add a new table from your source data store to the data flow, drag the desired table from the Object Gallery to the graphical editing pane.

   Note: The Object Gallery shows all external tables in the source data store that you have selected in the Show Tables From drop-down list box. If a table in your source-system database is not an external table in your source data store, then you must import that table (see "Importing External Tables" on page 216) if you wish to specify it available to the graphical extraction-command builder.

   You can import external tables from the Extraction Command dialog box by selecting Import External Tables from the Action menu.

6 To remove duplicate rows of data from your output (that is, to perform a SELECT DISTINCT operation), check **Remove Duplicate Rows from Output Data**.

7 To display joins, as well as data flow, check **Show joins in addition to data flow**.

Using SQL Expression Builder

You can use the SQL Expression Builder (see "Figure 90: SQL Expression Builder Dialog Box" on page 280) to specify transformations that are to be applied to your source data. The SQL expression builder is displayed whenever you choose to edit the data-flow between two elements in the **Data Flow** tab of the Extraction Command dialog box.

Figure 9-6: SQL Expression Builder Dialog Box

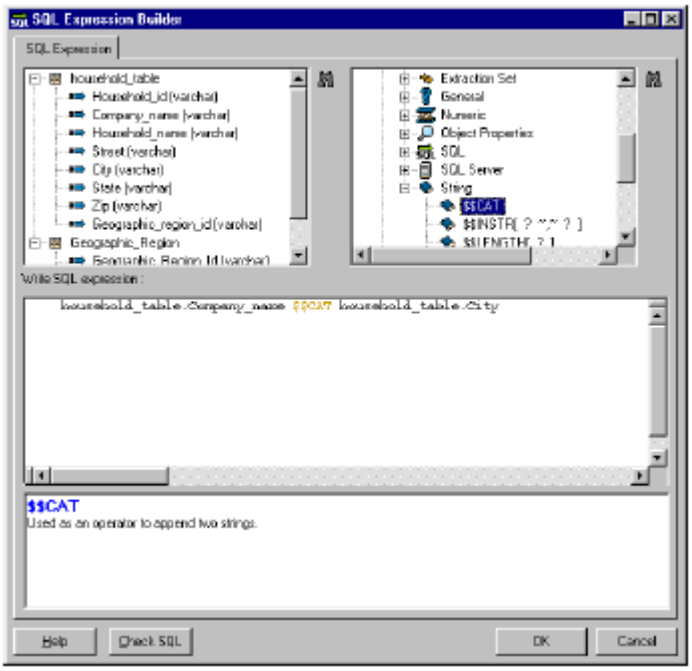


Figure 90: SQL Expression Builder Dialog Box

You build an expression in the center pane of the SQL Expression Builder. You can type directly in this pane, or you can add elements from the two upper panes.

1. The upper left pane shows all available columns in the source data-store tables. To add a column to the expression, double-click the name of the desired column. To search for a particular column, click **Find** .
2. The upper right pane shows all available functions. These functions are grouped into vendor-independent and platform-specific functions. Infor recommends that you use vendor-independent functions wherever possible.

   When you select a function in this pane, a description of the function is shown in the bottom pane. To add a function to your expression, double-click the name of the function. To search for a particular function, click **Find** .

   Note: Vendor-independent functions are hidden if a file datastore or an ODBC datasource that is not DB2, SQLServer, or Oracle is being used.

   When you have finished building your expression, click **Check** to verify that your expression is syntactically correct. If your expression is syntactically correct, click **OK** to complete the definition.

   Note: In order to ensure correctness, SQL syntax is verified using the database engine of the source system that you have specified in the General tab of the Extraction Command dialog box. Therefore, you must be connected to the source data store in order to check the syntax of your expression.

Defining Joins

The **Joins** tab of the Extraction Command dialog box (see "Figure 91: Extraction Command Dialog Box: Joins Tab" on page 281) allows you to configure how data from multiple source-system tables is combined when data is extracted. A dashed line between two source-table columns indicates that the values in these columns must be equal in order for two rows to be combined.

You can specify a new join criterion by dragging a line between two source-table columns, as you do when defining data flow (see "Defining New Data Flow" on page 278).

Figure 9-7: Extraction Command Dialog Box: Joins Tab

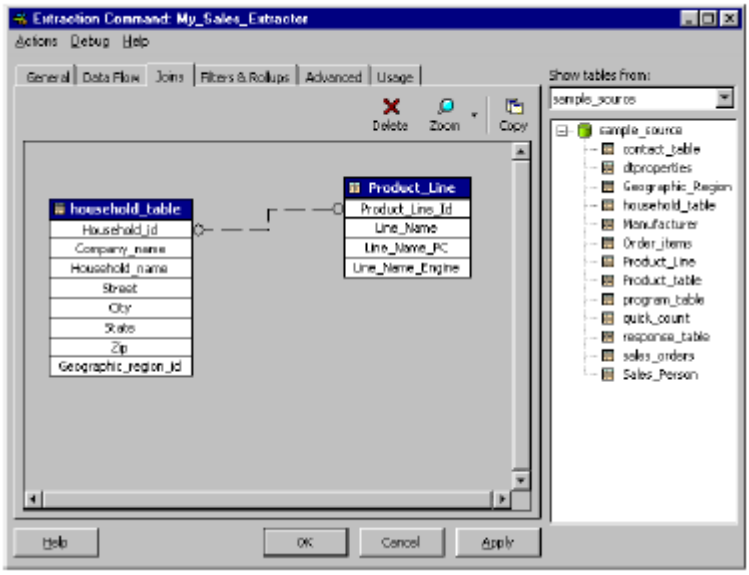


Figure 91: Extraction Command Dialog Box: Joins Tab

Defining Outer Joins

When you originally define a join between two columns, only data from table rows that have the same value in the joined columns is extracted (that is, an inner join is performed). You can also specify that all rows from one of the tables must be extracted, even if there is no matching value in the other table (that is, you can specify a right or left outer join).

To do this, right-click on the join line and select **Use All Rows From <Table>** from the pop-up menu, where **<Table>** is the table from which you wish to include all rows. To return to the original join criterion (inner join), select Do Not Join Null Rows from the pop-up menu.

Default Joins

You can specify that a join must be performed in every extraction command that uses the two joined tables (that is, you can specify a foreign-key relationship between the tables).

Note: When you create a default join, this join is used in all extraction commands that use both joined tables.

Specifying Default Joins

1. Select the join that you wish to apply to all relevant extraction commands.
2. Right-click the join.
3. Select **Make this a Default Join** from the pop-up menu. When you do this, a check mark appears next to **Default Join** in the menu.

   To remove a default join, right-click on the join and select **Default Join** to remove the check mark.

   Note: Definition of a default join takes effect immediately. You cannot delete the default join that you have defined by pressing **Cancel** in the Extraction Command dialog box.

Defining Filters and Rollups

The **Filters & Rollups** tab of the Extraction Command dialog box (see "Figure 92: Extraction Command Dialog Box: Filters & Rollups Tab" on page 283) allows you to set criteria that limit the data that is extracted from the source system or roll up results along selected columns.

The Additional Filters on Data Flow Pane

This pane displays filters on the data that you wish to extract from your source system.

1. To add a new filter, click **New** next to the Additional Filters on Data Flow pane. Clicking this button displays the SQL Expression Builder (see "Using SQL Expression Builder" on page 279), which you can use to define your desired filtering criterion.
2. To edit an existing filter, double-click it in the Additional Filters on Data Flow pane.

   When the extraction command is executed, all of the filters that you have defined here are applied to the source data before the staging table is populated.

Figure 9-8: Extraction Command Dialog Box: Filters & Rollups Tab

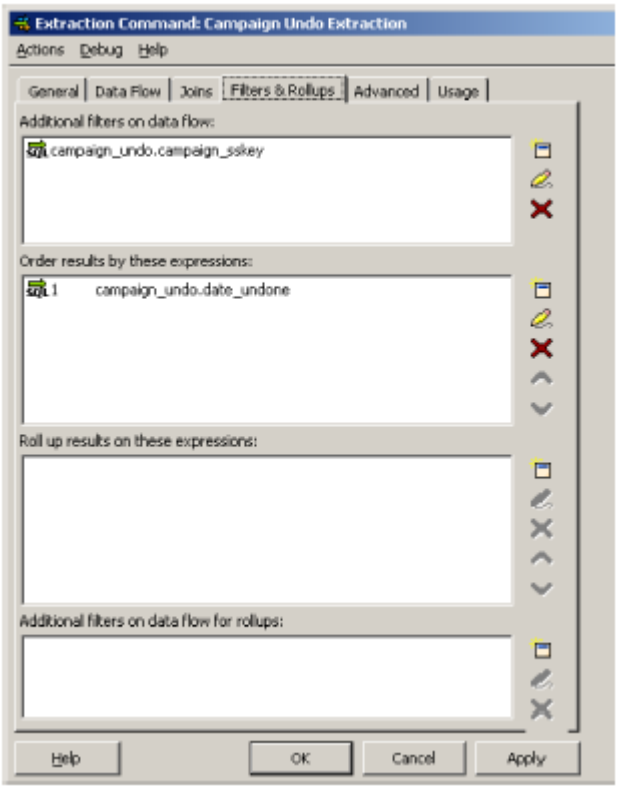


Figure 92: Extraction Command Dialog Box: Filters & Rollups Tab

The Order Results by These Expressions Pane

You can specify the order in which values are returned by an extraction command by adding expressions to the Order Results by these Expressions pane. You add and edit values in this pane in the same way that you do in the Additional Filters on Data Flow pane. Use the arrow buttons to order more than one expression. Expressions that are displayed at the top of the list have a higher precedence than those that are listed at the bottom.

The Roll Up Results on these Expressions Pane

If your extraction command uses aggregate functions (such as SUM), you specify the columns along which you wish to aggregate (that is, the columns in the GROUP BY clause) in the Roll up Results on these Expressions pane. Aggregate functions are computed for all values of the columns specified here.

You can add and edit values in this pane in the same way that you do in the Additional Filters on Data Flow pane.

The Additional Filters on Data Flow for Rollups Pane

If your extraction command uses aggregate functions (such as SUM), you can filter your results based on the values of these aggregate functions (that is, you can specify a HAVING clause) by adding expressions to the Additional Filters on Data Flow for Rollups pane.

You add and edit values in this pane in the same way that you do in the Additional Filters on Data Flow pane.

Using SQL to Define Extraction Commands

To define an extraction command with an SQL statement, select **Write SQL by Hand** in the **General** tab of the Extraction Command dialog box. When you select this option, the Extraction Command dialog box has an **SQL Statement** tab, in addition to the **General** , **Advanced** and **Usage** tabs.

Note: If you choose to define an extraction command with an SQL statement, you not edit that command with the graphical extraction command builder after you have specified changes to the SQL code.

EpiChannel automatically resolves SQL dialect problems if you use the Infor Campaign Management database-vendor independent macros consistently in your SQL. (See "Appendix A, “Infor Campaign Management Macros,”" on page 441 for more information.) If you do not use these macros, then the SQL code must be in the dialect of the database engine of the input data store.

To define the extraction step, enter the appropriate SQL code in the **SQL** tab of the dialog box. The code in this pane is automatically colored based on SQL syntax. See "Extraction Utility Steps" on page 298, for details of the required SQL code.

In most cases, it is best to start with the built-in SQL template for the table that you are populating, as described in the next section.

Displaying Sample SQL

If you choose to write SQL by hand and if you have selected a table for a SQL statement to populate, then you can click **Template** in the**SQL Statement** tab of the Extraction Command dialog box to append sample SQL code based on the structure of the destination table.

This sample code is in the form of a code template that you can modify based on the structure of your source system. The SQL template lists all of the columns that must be returned by the statement.

You must modify this template to contain the FROM and WHERE clauses appropriate for the tables in the source system and to place the proper data in each column. The statement must return all columns required by the destination table, but these columns can be returned in any order. If the statement returns any additional columns that are not found in the destination table, then those columns are simply discarded.

You can also view sample template code for a single column in a table:

- Click the small arrow to the right of the **Template** button to display a drop-down menu that contains all of the columns in the target table.

- Select the column template code you want to display. Select All to display code for all of the table's columns.

To view the macro's translation for your server, select your server type from the drop-down list box at the bottom of the dialog box and click **View Translation**. The translation for the selected database server is displayed in the SQL Results window.

Base Dimension Staging SQL Statements

Base dimensions are the physical dimension tables in the EpiMart database. As described in "Defining the EpiCenter Schema," on page 85 you use Admin Manager to configure the dimension columns for each base dimension table. During an extraction job, one or more SQL statements can be executed against the base dimension staging table to populate these columns.

The purpose of the base dimension staging query is to extract the latest values from the source system. You need not be concerned with determining when a value has changed, because the Infor Campaign Management semantic templates perform this task. You can, however, speed up the extraction process by using the Infor Campaign Management extraction macros (see "Extraction Macros" on page 480 and "How EpiChannel Identifies Data To Be Extracted" on page 362) to avoid extracting data that you know has not changed since the last extraction.

SQL Statement Template for Base Dimensions

As described in "Displaying Sample SQL" on page 284, you can use the **Template** feature in the **SQL Statement** tab of the Extraction Command dialog box to provide an SQL template for a given dimension table. For example, assume you define a base dimension called **Customer** with the following dimension columns:

- `FullName`
- `Age`
- `City`
- `State`
- `ZipCode`

In the Extraction Command dialog box, select the option **Populates dimension table** and choose the dimension table (**Customer**) from the drop-down list box. In the **SQL Statement** tab, click **Template** to display the following SQL statement in the dialog box:

```
SELECT

<YOUR EXPRESSION> customer_sskey,

$$TO_EPIDATE[ $$DBNOW ] date_modified,
```

```
$$NVL[ <YOUR EXPRESSION> ~,~ 'UNKNOWN' ] FullName,

$$NVL[ <YOUR EXPRESSION> ~,~ 'UNKNOWN' ] Age,

$$NVL[ <YOUR EXPRESSION> ~,~ 'UNKNOWN' ] City,

$$NVL[ <YOUR EXPRESSION> ~,~ 'UNKNOWN' ] State,

$$NVL[ <YOUR EXPRESSION> ~,~ 'UNKNOWN' ] ZipCode

FROM

<YOUR TABLE>
```

Note: The displayed template code also has a comment for each column that indicates the physical type of that column.

This is a regular SQL statement for which you must provide the values `<YOUR EXPRESSION>` and `<YOUR TABLE>`. You must assign each column in the `SELECT` list a valid SQL expression for the value of its destination column. A `FROM` clause is required to specify this a valid statement; a `WHERE` clause is optional.

Unknown Values

Null values are not allowed in any field of the staging table. This requirement is necessary because Infor Campaign Management uses `GROUP BY` statements at end-user query time to form the tables and charts of front-end applications. However, fact rows that aggregate on columns with null values are left out of the resulting reports because nulls are removed from `GROUP BY` results. Since checking for such null values significantly increases query-response time, null values are not permitted in EpiMart fact and dimension tables.

Since null values are not permitted in the staging table, you must substitute an actual value for all null values from your source system. In most cases, Infor recommends that you substitute the string `'UNKNOWN'` for any null values using the `$$NVL` macro, as shown in the template. The Infor Campaign Management schema generator generates an `UNKNOWN` row in every dimension table. By convention, the string `'UNKNOWN'` is used to refer to this row. If `'UNKNOWN'` is a valid key value in your dimension table, you can use any other string that is not used as a key value in the dimension table. Infor Campaign Management semantics automatically map all unmatched foreign keys in the fact staging table to the `UNKNOWN` row.

Template Columns

The columns displayed in the template include the dimension columns that are defined in the Base Dimension dialog box, as well as two implicit columns: `customer_sskey` and `date_modified`. These additional columns are added to every base dimension table by the Admin Manager Adaptive Schema Generator and must be populated by the staging SQL.

The SSKey Column

The first column is a source system key (`sskey`) and must be unique for every element of the dimension. The concept of an `sskey` is important in an EpiMart because the semantic templates use the `sskey` value to determine whether a row in the staging table represents the same source system entity as a row that already exists in the dimension table. The `sskey` is a variable length string, normally of maximum length 50, or an integer. It corresponds to the primary key of the source system table or the tables that specify this query.

Note: The sskey value is not directly available for Web-page configuration. If the sskey column is of interest to end users for querying, then Infor recommends that you create an additional dimension column that you populate with the same values as the sskey column.

The Date_Modified Column

The `date_modified` column is added to all base-dimension staging tables and is used to identify when a base dimension row is inserted into the EpiMart database. If the source system contains a **creation date** field, then this field must be used. Otherwise, you can use the source system's expression for "right now," which causes the `date_modified` column of newly extracted rows to assume the date on which they are extracted into the EpiMart database. Infor recommends that you use the `$$DBNOW` macro, which is automatically expanded to the appropriate expression for the current date.

Dates must be returned as strings, for example, `'5/26/1999'`. This can be completed using the `$$TO_EPIDATE` macro, as shown in the template.

Remaining Dimension Columns

The remaining columns in the `SELECT` list correspond to the dimension columns that are defined in the Base Dimension dialog box. These columns must be populated using expressions that are appropriate for the meanings of those columns. Any SQL expression that can be executed against the source RDBMS is valid. Infor provides a set of SQL macros that are automatically expanded to the correct syntax for your source system. Use of these macros facilitates the cross-platform usage of your SQL statements. See "Appendix A, "Infor Campaign Management Macros,"" on page 441 for more information.

Binning

In dimension tables, source-system entries with large numbers of possible values are often classified into categories or ranges. For example, a source system can record the exact ages of individuals, while a dimension simply records the age range into which an individual falls. This kind of categorization is called binning, and the Infor Campaign Management `$$CBIN_*` and `$$NBIN_*` macros are designed to help you simplify this process. See "Appendix A, "Infor Campaign Management Macros,"" on page 441 for information on these macros.

Duplicate sskeys

If, during a single extraction, a staging table is loaded with two or more rows with the same `sskey` , then the last row specified is used. You can also configure custom semantic types that do not check for duplicates or that fail when more than one row with a given `sskey` is found.

See "Semantics," on page 223 for a description of semantic types.

Note: If you use a semantic type that does not check for duplicates, you must ensure that no duplicates are found in your extracted data.

Dimension Staging Queries with Joins

Infor Campaign Management allows the use of joins in base-dimension staging queries. Star schemas typically de-normalize data structures in transactional systems into flat hierarchies, and you must be aware of what the granularity of a base dimension represents in this circumstance.

For example, If you want to use a Cartesian product of two tables in a base dimension staging query, unless the `sskey` of the result set combines the primary keys of the two tables that are being crossed.

It is more common for a single table to “drive” the result set, with other tables joined through unique key lookups to provide additional textual values. For example, a Product Master table in the source system represents the driving table of a `Product` base dimension. In this case, the `sskey` value can be the primary key of the `Product Master` table. Other tables with textual values for such columns as `Product_Line` or `Platform` can be joined with this master table. In such a case, you must ensure that the joined columns of the lookup tables are properly indexed, usually with UNIQUE indexes.

Base Dimension Queries with DISTINCT Fact Values

Sometimes dimensions do not have a corresponding master table in the source system. For example, an **Order** fact can have an **Order Type** associated with it, with several possible values. These values are embedded directly in the rows of the source system’s **Order** table, but there is no lookup table that contains all possible values.

In this case, a `SELECT DISTINCT` query against the **Order Type** column of the Order table in the source system can be appropriate for populating base-dimension staging tables in the EpiMart database. The alternative to this method is the use of degenerate dimensions in the fact table. The disadvantage of using a degenerate dimension is that degenerate dimensions cannot be aggregated.

Populating Fusion Staging Tables

If you are fusing elements of a dimension table (see "Fusion and Fission" on page 263), you must specify fused elements for that dimension. You specify fused elements by populating the corresponding fusion staging table and then applying a Latest Dimension Value with Fusion/Fission semantic instance.

A fusion staging table has the following template code:

```
SELECT
<YOUR EXPRESSION> child_sskey,
<YOUR EXPRESSION> parent_sskey,
<YOUR EXPRESSION> fusion_flag
FROM
<YOUR TABLE>
```

When you fuse two dimension elements, one of the elements (the parent) is chosen as the representative of both elements. The data for the parent is used in all analytical queries instead of the data for the other fused element (the child).

When writing staging SQL code for fusing two elements, assign the `sskey` of the parent to the `parent_sskey` column and the `sskey` of the child to the `child_sskey` column. To fuse the pair of elements, set the value of `fusion_flag` to `1`. To fission a pair of elements that has previously been fused, set the value of `fusion_flag` to `0`.

Fact Staging SQL Statements

SQL statements that populate fact staging tables are generally more complex than the statements that are used to load dimension staging tables. As with base dimension tables, the columns of the `SELECT` statements are determined by the metadata definition of the fact table in addition to certain implicit rules.

SQL Statement Template for Facts

For example, you can define an Order fact with the following dimension roles:

- **CustomerBillTo**
- **Product**
- **CustomerShipTo**
- **SalesPerson**

This EpiCenter contains a single degenerate dimension called `OrderNumber` , and the Order table has two fact columns: `net_price` and `number_units` that represent the extended amount for an order line item, along with the quantity.

Click **Template** in the **SQL Statement** tab of the Extraction Command dialog box (with the **Populates Fact Table** option selected and the `Order` table selected in the drop-down list) displays the following SQL in the dialog box:

```
SELECT
<YOUR EXPRESSION> ss_key,
$$TO_EPIDATE[ <YOUR EXPRESSION> ] date_key,
<YOUR EXPRESSION> transtype_key,
<YOUR EXPRESSION> process_key,
$$NVL[<YOUR EXPRESSION>~,~'UNKNOWN'] customerbillto_sskey,
$$NVL[<YOUR EXPRESSION>~,~'UNKNOWN'] product_sskey,
$$NVL[<YOUR EXPRESSION>~,~'UNKNOWN'] customershipto_sskey,
$$NVL[<YOUR EXPRESSION>~,~'UNKNOWN'] salesperson_sskey,
$$NVL[<YOUR EXPRESSION>~,~'UNKNOWN'] ordernumber_key,
$$NVL[<YOUR EXPRESSION>~,~0] net_price,
$$NVL[<YOUR EXPRESSION>~,~0] number_units
FROM
<YOUR TABLE>
```

Note: The process_key column only appears in the template for fact tables that have the Contains Transactional and Statelike Rows option checked in the Fact dialog box.

As with base dimension staging queries, you must identify what a row in this fact table represents. In this example, a fact row indicates a line item of a sales order. The salesperson can get full credit for a line item, or the fact row can be a particular amount of credit that a salesperson received for an order line item. Typically, the `FROM` clause of this query joins the `Order Line Item` table to the `Order Header` table in the source system.

Template Columns

The columns in the `SELECT` list can now be divided into these categories:

- Implicit columns that are added automatically
- Dimension role foreign keys
- Degenerate dimension keys

- Fact numeric columns

The SS_Key Column

As with base dimensions, each fact staging row contains an `ss_key` (notice the difference in spelling--base dimensions use sskey rather than ss_key) that uniquely identifies this row in the source system. In this example, the `ss_key` can be a concatenation of the Order Number with the Order Line Number (since this combination is presumably unique). `ss_key` s is used on subsequent extractions to prevent duplicate copies of the fact row from being created in the EpiMart database.

The Date_key Column

The `date_key` indicates when the fact occurred. Since time is a central component of the EpiCenter, each fact table must contain this column. Many facts are time based; in this example, `date_key` can represent the time at which the last order is updated. However, if time is not important for this fact, then the current system time can be used as a placeholder.

Note that `date_key` is granular only to that single day when the fact occurred. For best results, the fact SQL Statement must return the day as a string, for instance, “5/1/1999.” Infor recommends that you use the `$$TO_EPIDATE` macro to ensure that dates are in the proper format.

The TransType_Key Column

The transaction type indicates the “slice” of the fact table to which the row belongs. For example, the Order fact can hold both Bookings and Shippings, and `transtype_key` identifies which fact staging rows are bookings and which are shippings. The SQL statement must return a numeric key that matches one of the transaction types defined in the Configuration dialog box in Admin Manager (see "The Transaction Types Tab" on page 149). See "Semantic Instance Extraction Steps" on page 187 for more information about `transtype_key` .

Note: Transaction type values in the range 10000-20000 are reserved for use by Infor.

The Process_Key Column

A fact staging table can contain different types of rows that need to be handled in different ways by the semantics. The `process_key` identifies rows from the fact table to be processed in a specific way by a semantic instance. A value of 1 indicates a transactional fact, and a value of 2 indicates a state-like fact. A semantic type that has the **Row Type** option set to **Transactional/Statelike** uses the value in the `process_key` column to distinguish between transactional and state-like facts. If you are using a semantic type (such as Transactional or Statelike) that does not have the **Row Type** option set to **Transactional/ Statelike** , then you do not need to include the `process_key` column.

Dimension Role Foreign Keys

For all dimension roles that are included in the dimensionality of the fact table, you must enter values for the corresponding dimension-role foreign keys. Notice that the names of the columns in the SQL template are `DimRoleName_sskey` , where DimRoleName is the name of a dimension role that is

included in the dimensionality of the fact table. The values in these fields refer to `sskey` s in the base dimension tables that correspond to the dimension roles.

You need to understand the meaning of each base dimension table to ensure that these keys resolve properly. If the `sskey` of the Product base dimension is taken from a **Product Master** list in the source system, then `product_sskey` in the Order table must also refer to an entry in the **Product Master** list. If a base dimension is the cross-product of two source system tables, the fact staging keys for that dimension must also represent a unique cross-product entry.

Degenerate keys in the fact staging query must be populated with string values. In the example above, the `ordernumber_key` field can be populated with an order number value from the Order Header table, such as 253AD56.

Note: If you have defined additional dimension roles that refer to the date dimension, then the names of the foreign keys for those roles end in key rather than sskey. For example, if you have defined a dimension role called inquiry_date that refers to the date base dimension, then the name of the foreign key for that dimension role is inquiry_date_key.

Additional Fact Table Columns

Finally, the numeric columns represent the actual quantities and raw amounts that are associated with each fact entry. In general, each column must be an additive amount, in order to ensure correct front-end query results. For example, total dollar amounts for line items can be added, but unit prices cannot be meaningfully added across fact rows.

Using External Tables as Inputs to Staging Queries

Sometimes it can be necessary to bring data into EpiMart external (temporary) tables before performing any joins. For example, this is necessary if the source system's SQL limits your ability to manipulate the data. The full power of the EpiMart database's RDBMS engine can then be used to load the staging tables. In this case, the following sequence of actions is usually employed:

- Drop any indexes on the external tables for fast loading into EpiMart external tables.
- Load the external tables from the source system into external tables.
- Create any indexes needed for fast joins on the external tables.
- Load the staging tables using queries against the EpiMart External tables. All query plans must use the indexes built in Step 3.

Note: This technique can also be used to work around database driver limitations. For example, if you are extracting BIGINT data from an ODBC data source using an ODBC driver that does not support the BIGINT datatype, you can convert the BIGINT data to VARCHAR in an extraction command that extracts to an external table, and then convert back to a BIGINT when extracting from the external table to a staging table.

Note that you can also be able to use pre-staging tables rather than external tables when you wish to manipulate data before loading staging tables. See "Pre-Staging Tables" on page 185 for more information.

Extracting International Data

Localized versions of Infor Campaign Management can extract data from source systems that use any supported language and locale. The supported code pages for source systems are shown in "Supported Source-System Code Pages" on page 293.

Table 56: Supported Source-System Code Pages

Code Page Type	DB2	Oracle	SQL Server
Latin 1	8859_1	ISO-8859-1	CP1252
Latin 5	ISO-8859-9	CP1254	
Unicode	UTF-8	UTF-8	UCS-2
Japanese	UTF-8IBM-eucJP	Shift JIS EUC (JA16SJIS)	CP 932

Note: While EpiMart table and column names must use the ASCII character set, source-system tables and columns can use characters from the source system's character set.

If the source system uses a different code page from the EpiMart database, then EpiChannel translates data from the source-system code page to the EpiMart code page wherever possible. When extracting data, be sure that all characters found in the extracted data can be translated to characters in the code page used by your EpiMart database. For example, Japanese characters from the EUC code page cannot be translated to characters in the Latin 1 code page, but ASCII characters from the EUC code page can be translated to the equivalent characters in the Latin 1 code page.

If your source-system data contains untranslatable characters (that is, characters that are not included in the character set of the source system's code page or that cannot be translated to characters in the EpiMart code page), then EpiChannel translates those characters to the `?` (question mark) character.

Note: If your source system uses a multi-byte character set, then EpiChannel can not be able to determine the length of an untranslatable character. For this reason, characters appearing after an untranslatable character in source-system data can be incorrectly translated.

Character data in SQL Server EpiMarts is stored in NCHAR and NVARCHAR columns. Character data in Oracle and DB2 EpiMart databases is stored in CHAR or VARCHAR columns, but extraction from NCHAR and NVARCHAR columns is supported.

Extracting NCHAR and NVARCHAR2 Data on Oracle

If your EpiMart database resides on an Oracle database server, you must ensure that string literals that are used in expressions with NCHAR or NVARCHAR2 source-system columns are in the appropriate national format. You can do this by prepending the `N` operator to the literal.

For example, to substitute the value `'UNKNOWN'` for null values in an NVARCHAR2 source-system column called `my_nat_col`, you might use the following SQL code:

```
SELECT
...
$$NVL[my_nat_col ~,~ N'UNKNOWN'] my_dest_col
FROM
my_source_table
```

Here `my_dest_col` is a VARCHAR column in the destination table and `my_source_table` is the source table.

Localized Log Files

EpiChannel log files can be written in any language that is installed with your EpiCenter. To specify the logfile language, use the **Languages and Locales** tab of the Configuration dialog box in Admin Manager. See "The Languages and Locales Tab" on page 158 for information.

Extraction Groups

An extraction group is an ordered sequence of extraction steps. To define a new global extraction group, take the following steps.

Defining a Global Extraction Group

1. In the Extraction Steps dialog box, select the group to which you wish to add your new extraction group and create a new **Group**, as described in "The Extraction Steps Dialog Box" on page 269
2. In the Global Extraction Group dialog box, enter a name, developer label, and, if desired, a description for the group.
3. Indicate the action to be taken On Error (see "Table: Possible Actions On Error, on page 9-42" on page 301).
4. Click **OK** to create the group. The new group is displayed as an object in the dialog box. The icon for a global extraction group is a beige folder with a globe.

After you have created an extraction group, you can add extraction steps to that group. A group can contain other groups.

Note the following:

- To change the order of extraction steps within an extraction group, select an extraction step in the Extraction Steps dialog box and click **Up** or **Down** to move it to the desired location in the list.
- To move an extraction step to a different extraction group, drag the extraction step to the desired group.

Semantic Instances

A semantic instance is an extraction step that applies a semantic template to a specific data mart table. Semantic instances process data found in staging tables, and load the results into the appropriate inactive A/B data mart table (see "Data Mart Mirroring: A and B Tables" on page 191).

Whenever you make a change to your EpiCenter configuration that must be reflected in the data mart tables (such as changing an aggregate definition), you must run a semantic instance on the changed table in order to implement your changes.

See "Semantics," on page 223 for more information about semantics.

Defining a Semantic Instance

1. In the Extraction Steps dialog box, select the group to which you wish to add your new semantic instance and create a new **Semantic Instance**, as described in "The Extraction Steps Dialog Box" on page 269.
2. In the Semantic Instance dialog box (see "Figure 93: Semantic Instance Dialog Box" on page 296), select the type of table that the semantic instance **References** (that is, the type of table to which the semantic instance is being applied). A semantic instance can reference a **Dimension Table** or a **Fact Table**.

Figure 9-9: Semantic Instance Dialog Box

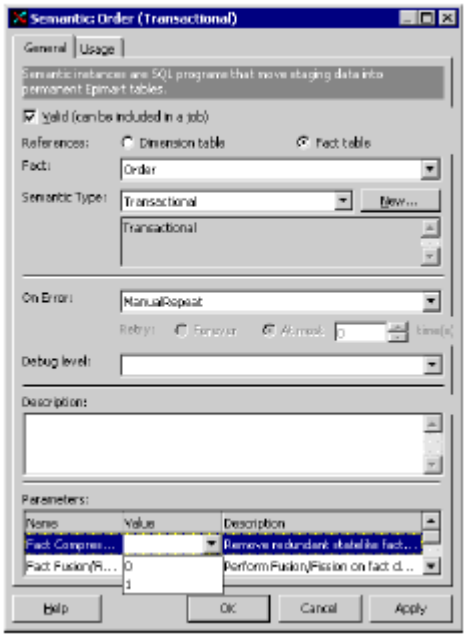


Figure 93: Semantic Instance Dialog Box

3 Select the associated table name from the drop-down list box. The label of the table-name list box is **Fact** or **Dimension** , depending on the type of table that you are referencing.

4 Select the semantic type from the drop-down list box. See "Dimension Semantic Types" on page 224 and "Fact Semantic Types" on page 238 for descriptions of semantic types.

Click **New** if you wish to define a new semantic type (see "Defining New Semantic Types" on page 266).

5 Indicate the action to be taken **On Error** (see "Table: Possible Actions On Error, on page 9-42" on page 301).

6 Enter a **Description** , if desired.

7 If appropriate, set the values of the **Fact Fusion** , **Fact Compression** , or **Dimension Data Delete** parameters. See "Fusion and Fission" on page 263 for a discussion of fusion. See "Fact Table Compression" on page 254 for a discussion of fact compression. See "Dimension Data Deletion" on page 234 for a discussion of dimension data deletion.

Note: The Fact Fusion and Fact Compression options have no effect on dimension semantics. The Dimension Data Delete option has no effect on fact semantics.

8 Click **OK** to define the semantic instance.

System Calls

As mentioned, sites with more complex databases requires multiple stages and additional commands, such as commands to create lookup tables, gather data into ranges (see "Base Dimension Staging SQL Statements" on page 285), and detect duplicates. For this purpose, you can use system calls, which are executed during a job as if invoked from the console command line.

Defining a System Call Object

1. In the Extraction Steps dialog box, right-click the global extraction group to which you wish to add the system-call object. By default, some pre-defined system calls are available in the **System Call** global extraction group, but you can create system calls in any location.
2. In the pop-up menu, choose **System Call** from the New submenu to display the System Call dialog box.

Figure 9-10: System Call Dialog Box

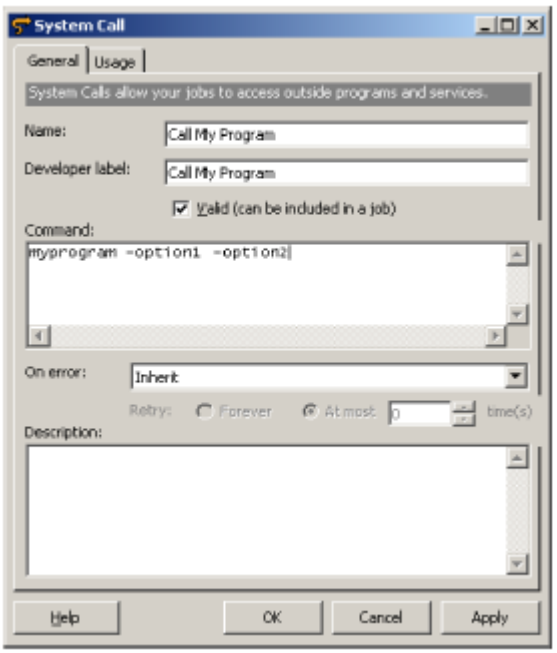


Figure 94: System Call Dialog Box

3. Enter the name and developer label for the system call, the action to be taken upon error (see "Table: Possible Actions On Error on page 9-42" on page 301), the command line for the system call, and, optionally, a description.

   Any output from a system call job step is redirected to a Syscalls subdirectory of the logging directory currently being used by EpiChannel.

   Note: Infor recommends against using remote shell (rsh) commands as system calls. The rsh command returns 0 if it succeeds in starting a remote shell. It does not return the return status of the remote process. Thus, you cannot monitor the success or failure of a command invoked by rsh.

Extraction Utility Steps

The following extraction utility steps can be defined for your EpiCenter:

- AggBuild
- Cancel
- Cancel Backfeed
- Commit
- Commit Backfeed
- Create Current Views
- Dimension Backfeed Semantics
- Fact Backfeed Semantics
- MomBuild
- Obtain Backfeed Lock
- Refresh
- Release Backfeed Lock
- Rolloff
- Rolloff Backfeed Semantics
- Truncation

These extraction steps are described in "Extraction Utilities" on page 187. You can create any of these steps by right-clicking the global extraction group to which you wish to add the step and then selecting the desired type of extraction step from the **New** submenu.

Most of these extraction steps need no configuration beyond that required for any extraction step. The following sections describe the parameters available for configurable extraction utility steps. These parameters are all configured using a grid control in the extraction step dialog box.

Note: The Create Current Views extraction step fails for a given view if the view is in use.

AggBuild

"AggBuild Parameters" on page 298 shows the parameters that can be configured for an AggBuild job step.

Table 57: AggBuild Parameters

Parameter	Description
Force Restart	A value of 1 indicates that Agg Builder must ignore previous execution history and build all aggregates specified in the metadata
Rebuild Fact Aggs	A value of 1 indicates that Agg Builder must rebuild all fact aggregates from scratch. Fact aggregates are not incrementally updated.

MomBuild

The "MomBuild Parameters" on page 299 table shows the parameters that can be configured for a MomBuild job step.

Table 58: MomBuild Parameters

Parameter	Description
Force Rebuild	A value of 1 indicates that Momentum Builder must rebuild all accelerators from scratch. Accelerators are not incrementally updated.
Force Restart	A value of 1 indicates that Momentum Builder must ignore previous execution history and build all accelerators specified in the metadata
Minimum Query Parallel Degree	The value specifies a minimum level of parallelism for the database server to use when it creates an index. Currently, this setting applies to the Oracle server only, whose index creation statement allows for a parallel degree specification. See the discussion for Num Parallel Threads for additional details.
No Mirror	A value of 1 indicates that Momentum Builder must build accelerators for the Current tables rather than the Next tables.
Num Parallel Threads	The values sets the number of parallel threads to use when building accelerators. In building accelerators, each EpiChannel client thread sends a request to the database server to create an index or a table. MomBuilder automatically specifies the amount of parallelism to be used by the database server when it builds an index, as allowed by the database server. (Currently, only Oracle supports this feature for building indexes.) MomBuilder uses the value specified by the Minimum Query Parallel Degree parameter as a minimum degree of parallelism when Oracle creates the index. If MomBuilder finds that the requested number of parallel threads is much higher than the number of indexes to be built, it adjusts the database server's parallelism for building a single index to a higher degree. The total index building parallelism is the MomBuilder Num Parallel Threads setting times the degree of parallelism for each one index as determined by MomBuilder, with this latter value con-

Parameter	Description
	strained to a minimum value specified by Minimum Query Parallel Degree. **Note:** DB2 EpiMarts support multiple parallel threads.

Refresh

"Refresh Parameters" on page 300 shows the parameters that can be configured for a Refresh job step.

Table 59: Refresh Parameters

Parameter	Description
Infor Campaign Management Server Host	The name of the host machine on which the Infor Campaign Management Server is running.
Infor Campaign Management Server Port	The port on which the Infor Campaign Management Server is listening.
Infor Campaign Management Server URL	The URL of the Infor Campaign Management Server.
Use HTTP?	A value of 1 indicates that the refresh request must be made through an HTTP request. A value of 0 indicates that the refresh request must be specified through a direct socket connection to the Infor Campaign Management Server.

If values are not specified for any of these parameters, then the default values specified in metadata or in the Registry are used.

Truncation

A truncation step truncates a specified dimension staging table, fact staging table, or external table. Typically, extraction commands that populate a staging table or an external table are included in an extraction group with a truncation step for that table (see "Extraction Groups" on page 294).

In the Truncation dialog box, you must select the kind of table that you wish to **Truncate**. Select **Dimension Table** to truncate a dimension staging table, **Fact Table** to truncate a fact staging table, or **External Table** to truncate an external table. Then select the name of the table to be truncated from the drop-down list box.

Note: A truncation step for a fact or dimension table also truncates the inactive base table, in addition to the staging table, if it is used in a job in which the table is being populated with an extraction command that is marked as Initial Load. (See "The Load Options Tab" on page 309 for details.)

Configuring Extraction Steps for Restartability

If an extraction job fails, you can restart the job manually or EpiChannel can restart the job automatically. In this way, you can configure a job so that it is not necessary to repeat those parts of the job that have already completed successfully.

If an extraction step cannot be repeated without compromising data integrity, then include that step in the smallest possible extraction group that can be safely repeated. For example, if an extraction command that populates a staging table fails, you do not want to restart the job at that step. The extraction command have some data in the staging table, and any data that is extracted when the step is repeated is appended to the data that is placed in the staging table in the failed extraction.

Local extraction steps are used primarily to assemble global extraction groups in the sequence needed. To ensure restart-ability, always use global extraction groups (each of which is restartable) to assemble your local extraction step. (Each global extraction group—in this context—typically extracts or loads a single table.)

For most installations, Infor recommends that when you populate a table, you create an extraction group that contains a truncation step for that table followed by the extraction commands that populate the table as shown in "Figure 95: Typical Configuration for Table Population" on page 301.

In this example, the **On Error** property of the extraction commands is set to **Inherit** and the **On Error** property of the extraction group is set to **AutoRepeat**. Thus, if one of the extraction commands fails, the job is restarted at the beginning of the extraction group, ensuring that the table is truncated before the extraction commands are repeated.

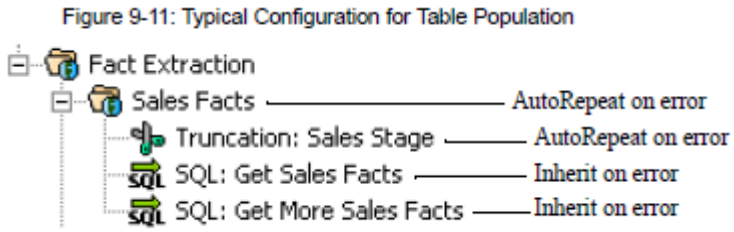


Figure 95: Typical Configuration for Table Population

"Possible Actions On Error" on page 301 lists the possible actions that EpiChannel can take if an error occurs.

Table 60: Possible Actions On Error

Action	Description of Action Taken on Error
Abort	Abort the entire extraction job. The job cannot be restarted automatically. If the job is restarted manually, then it restarts from the beginning of the job
AutoRepeat	Automatically retry the job step or job group, up to the maximum number of repetitions specified. If the maximum number of repetitions are reached, abort the job rather than retrying the step again. If

Action	Description of Action Taken on Error
	the job is restarted automatically or manually after aborting, then it restarts at the beginning of the current node.
Ignore	Ignore the error and continue.
Inherit	Abort the job step. The parent node of the group for this job step handles the error condition in accordance with its On Error setting. For example—using "Figure 95: Typical Configuration for Table Population" on page 301— if the SQL: Get Sales Facts job step fails (currently set to Inherit On Error), then EpiChannel uses the On Error setting for the job group parent node to handle the error. In this case, EpiChannel automatically retries the group of job steps, as the Sales Facts node is set to AutoRepeat.
ManualRepeat	Abort the job. The job cannot be restarted automatically. If the job is restarted manually, then it restarts at the beginning of the current node
Print	Print an error message to the console and continue.
StartOver	Automatically retry the job from the beginning, up to the maximum number of repetitions specified. If the maximum number of repetitions are reached, abort the job. If the job is restarted automatically or manually after aborting, then it restarts from the beginning of the job.

Extraction Jobs

10

After you have configured your EpiCenter, you can configure the extraction jobs to populate your data mart. Extraction jobs are collections of extraction steps.

Defining a Job

The Job dialog box enables you to define a job, create and order its job steps, assign data stores and roles, and schedule it in a queue. You can define as many extraction jobs as are necessary for your EpiCenter. For most installations, Infor recommends that you base the jobs that you configure on the default job templates that are provided for you (see "Extraction Job Templates" on page 319).

Figure 10-1: Job Dialog Box: General Tab

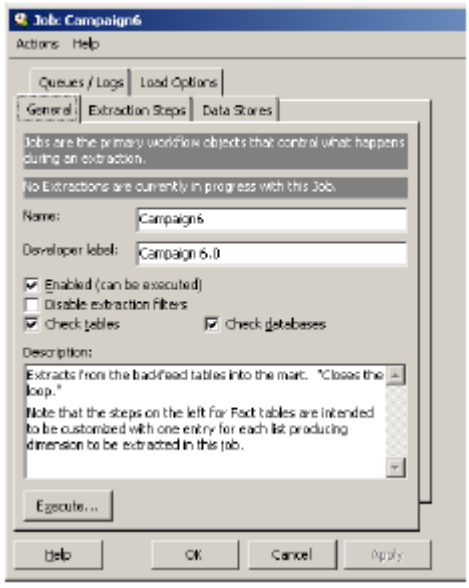


Figure 96: Job Dialog Box: General Tab

To open the Job dialog box for an existing job, double-click the icon for that job in the Jobs subfolder of the Extraction folder for your EpiCenter. To open a new job dialog box, right-click the Jobs folder and select **New Job**.

The Job dialog box has five tabs: **General** , **Extraction Steps** , **Data Stores** , **Load Options** , and **Queues/Logs**.

The General Tab

Use the **General** tab of the Job dialog box (see "Figure 96: Job Dialog Box: General Tab" on page 303) to define the following parameters:

- **Name** : The name of the extraction job as defined in EpiMeta.

  Note: Do not use multi-byte international characters in job names.

- **Developer label**: The name of the extraction job as displayed in Admin Manager.
- **Enabled (can be executed)** : Enables the job. EpiChannel executes only enabled jobs.
- New jobs are initially enabled. You can, however, disable a completely functional job in some circumstances to accommodate system changes. For example, if a database is in the process of being moved or repaired, jobs that extract from that database are disabled as a protection against accidental execution.

  Note: Extraction steps within a job can still remain disabled after a job is enabled if they are individually disabled. You can reenable them in the Extraction Steps pane of the Job dialog box.

- **Disable Extraction Filters**: Disable EpiChannel extraction filters for all job steps.

  EpiChannel has special extraction filters that are used for incremental extraction from a source database (see also, "Extraction Macros" on page 480). When the **Disable Extraction Filters** option is checked, EpiChannel ignores these extraction filters and retrieves all rows in the source databases that match the queries specified by the extraction commands in the job.

- **Check Tables**: Directs EpiChannel to check that all tables referenced by a job exist.

  Tables are checked at the beginning of a job. In most cases, you must leave this option checked.

- **Check Databases**: Directs EpiChannel to check that all databases referenced by a job exist.

  Databases are checked at the beginning of a job. In most cases, you must leave this option checked.

- **Description**: A textual description of the job.

Click **Execute** in the Job dialog box or select **Execute Job** from the **Actions** menu of the Job dialog box to display the Execute Job dialog box. This dialog box allows you to execute the job with EpiChannel. See "Executing a Job" on page 312, and "Running Jobs with EpiChannel" on page 351 for more information.

The Extraction Steps Tab

Use the **Extraction Steps** tab of the Job dialog box (see "Figure 97: Job Dialog Box: Extraction Steps Tab" on page 305) to configure and group the extraction steps that are used in extraction jobs.

You define the actual steps of a job by selecting objects in the Object Gallery and dragging them onto local groups in the left pane. Every extraction job automatically includes a root-level local group that has the same name as the job.

You can define the global steps that are used in extraction jobs in the Extraction Steps dialog box (see "Figure 85: Extraction Steps Dialog Box" on page 270), or in the Object Gallery pane of the **Extraction Steps** tab of the Job dialog box. See "Extraction Steps" on page 269 for information about defining individual extraction steps.

Note: Any changes that you make to a global extraction step in the Object Gallery affect all jobs that use that global extraction step.

Figure 10-2: Job Dialog Box: Extraction Steps Tab

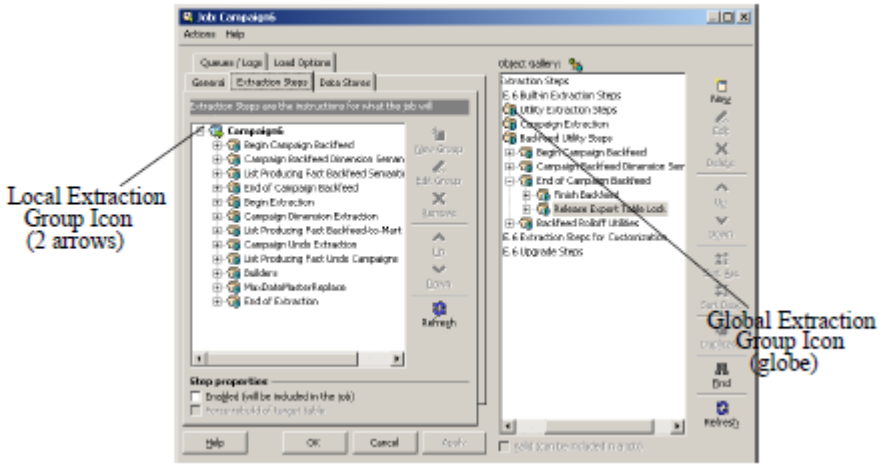


Figure 97: Job Dialog Box: Extraction Steps Tab

You can make the following changes in the **Extraction Steps** tab of the Job dialog box:

1. Define a new local group by right-clicking the local group to which you wish to add your new group in the left pane and selecting **New Local Group** . You can add a new job step to the root-level local group or to any local group that you have already defined. In the **Local Extraction Group** dialog box, enter a label name for the group and, if desired, a description. Click **OK** .
2. Local extraction groups are displayed in boldface, and the icon for a local extraction group is a blue folder with two arrows. Keep in mind that these groups are local to an extraction job. Global extraction steps, including global extraction groups, that are included in the job are not displayed in boldface.
3. To add a global extraction step to a job, drag the step from the Object Gallery onto the local group to which you wish to add it. You can add an extraction step to the root-level group, which has the same name as the extraction job, or to any local group that you have defined.
4. You can change the execution order of job steps. Order steps by selecting a step that you want to move and click **Up** or **Down** until the step is in the desired location. If your job includes a global extraction group, you cannot change the order in which the steps in that group are executed, except

by changing the global properties of the extraction group in the Object Gallery or in the Extraction Steps dialog box.

5 You can disable a job step that you have added to your job by selecting the step and deselecting the **Enabled (will be included in job)** check box under **Step Properties** . When you disable a step, the step is displayed with a red **X** over its icon. Instances of the extraction step in other jobs remain enabled.

6 You can also disable all instances of an extraction step in your EpiCenter by selecting the extraction step in the **Object Gallery** and deselecting the **Valid (can be included in job)** option. The disabled extraction step is greyed out with a yellow exclamation point.

7 To indicate that a semantic instance must not attempt to incrementally revise a fact or dimension table (see "Swapping Mirrored Tables" on page 192), select the semantic instance in the left pane and check the **Force Rebuild of Target Table** check box under **Step Properties** .

Note: When you use a Streaming semantic instance in an extraction job, the Force rebuild of target table option in the Extraction Steps tab of the Job dialog box is ignored during execution of the semantic.

8 To remove a step from a job, select the step in the left pane and click **Remove** . You cannot remove an extraction step that is in a global extraction group.

9 To edit a local extraction group, select the group and click **Edit** . You can select the group from the left pane or the **Object Gallery** .

Note: A Local Extraction Group icon consists of a blue folder with two arrows. A Global Extraction Group icon consists of a beige folder with a globe. (See "Figure 97: Job Dialog Box: Extraction Steps Tab" on page 305.)

Normal Extraction Order

Infor's recommended sequence of events during an extraction job is as follows:

1 Run **Begin Extraction** .
2 Execute base dimension extraction commands.
3 Execute base dimension semantic instances.
4 Execute fact extraction commands.
5 Execute fact semantic instances.
6 Run Agg Builder.
7 Run Momentum Builder.
8 Run **MaxDateMasterReplace** .
9 Run **End of Extraction** .

Dimension data must be extracted before fact data and base dimension semantic instances must be executed before fact semantic instances because:

- Fact-table entries make use of the numeric dimension keys that are generated by dimension semantic instances.
- EpiChannel uses the dimension-mapping information generated by dimension semantics during fact extraction.

- Streaming fact extraction steps require that the information generated by dimension semantics be stored in EpiChannel memory.

  Note: If an extraction job includes a dimension semantic that changes dimension values, such as Latest Dimension Value, then that job must also include related fact semantics. In particular, if a fact table includes an aggregate instruction that makes use of the dimension that is being changed, then the job must include a semantic instance for that fact table.

Time-Based Extraction Filters

You must use time-based extraction filters when extracting from a live source system (that is, a source system that changes during extraction) in order to ensure that all dimension data that is referenced by fact data is properly extracted.

If EpiChannel is extracting from a live source system, then new dimension and fact rows are created while the extraction occurs. For example, an order for a new customer appears in the source system after data for the **Customer** dimensions are extracted, but before **Order** data is extracted. Without time-based extraction filtering, the data for the new order is extracted into the fact staging table, but the data for the new customers are not extracted into the dimension staging table. Since the customer referenced in the order is not found in the base dimension table or the dimension staging table, the order is mapped to the **UNKNOWN** row of the **Customer** dimension.

You can use time-based extraction filters to ensure that new data that appears in the source system after extraction begins is not extracted in the current extraction job. See "How EpiChannel Identifies Data To Be Extracted" on page 362, and "Configuring an Extraction Command" on page 273 for more information on time-based extraction filters).

Note: If you are extracting from a live source system, Infor recommends that you use time-based extraction filters in your extraction commands. If you do not use such extraction filters, changes in your source system during extraction results in fact data that does not properly join to the associated dimension data.

Note: Time-based extraction filters can only be used when the data source is the same type of database (Oracle, SQL Server, DB2) as the EpiMart. They cannot be used for ODBC data sources.

Alternate Extraction Order for Non-Memory-Mapped Dimensions

If you are not using streaming extraction and none of your base dimensions can be mapped in EpiChannel memory (that is, none of your base dimensions have the **Allow EpiChannel to map SSKeys to Keys in Memory** check box checked), then you can use the following alternative extraction order:

1. Run **Begin Extraction** .
2. Load fact staging tables.
3. Load base dimension staging tables.
4. Execute base dimension semantic instances.
5. Execute fact semantic instances.

6 Run Agg Builder.

7 Run Momentum Builder.

8 Run MaxDate

9 Run **End of Extraction** .

When you use this extraction order, you do not need to use time-based extraction filters, even if you are extracting from a live source system. Extracting dimensions after facts ensures that all dimensions referenced by a fact are available for extraction. In the example above, if an order for a new customer appears after **Order** data is extracted, but before **Customer** data is extracted, then only the data for the new customer is added to the EpiCenter in this extraction. When the data for the new order is added in the next extraction, it is correctly joined to the customer data that has already been extracted.

The Data Stores Tab

The **Data Stores** tab of the Job dialog box (see "Figure 98: Job Dialog Box: Data Stores Tab" on page 309) allows you to assign the data stores and data store roles to a job and its individual job steps. Default data stores and data store roles and any new data stores that you have defined appear in the **Object Gallery**. See "Data Stores" on page 203 for more information about data stores.

Data Stores

Assign a data store by selecting a data store object from the Object Gallery in the right window of the dialog box and dragging it onto the desired data store role in the left pane. The object becomes attached. For example, select **EpiMart** in the Object Gallery and drag it onto the **Output** role for the top-level job step.

Figure 10-3: Job Dialog Box: Data Stores Tab

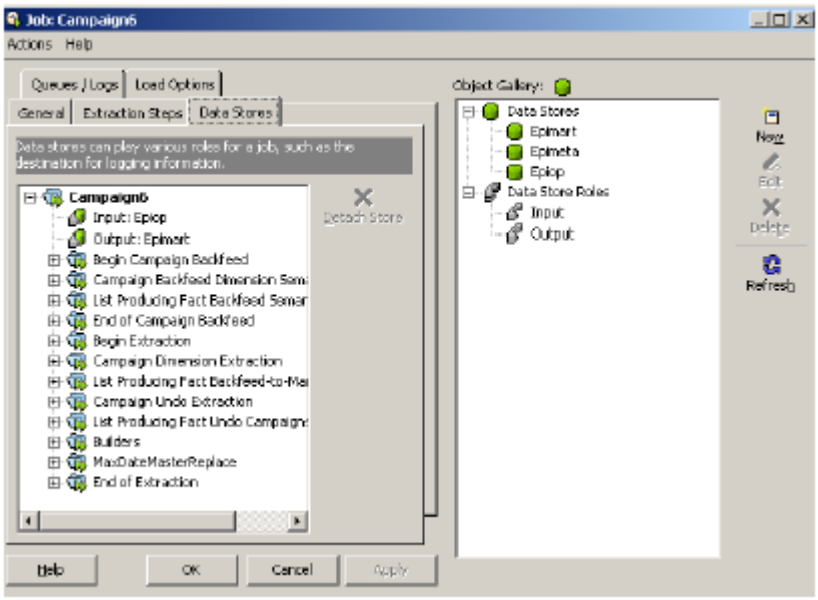


Figure 98: Job Dialog Box: Data Stores Tab

If a job step does not have a data store assigned to a given role, then that step inherits the assignment from its parent. If the parent does not have a data store role assigned, then the step inherits the assignment from the closest ancestor that does have a role assigned.

To remove an attached object, select it in the left window of the dialog box and click **Detach Store** .

Data Store Roles

You can define custom data store roles, which you can then use in system calls. To create a new data store role, right-click in the Object Gallery and select **New Data Store Role** , or click the New icon and select **Data Store Role**.

To delete a user-defined data store role, right-click on the data store role and choose **Delete** .

Note: You cannot delete the built-in data store roles.

The Load Options Tab

Use the **Load Options** tab of the Job dialog box to perform an initial load or streaming load for any table in your data mart.

Figure 10-4: Job Dialog Box: Load Options Tab

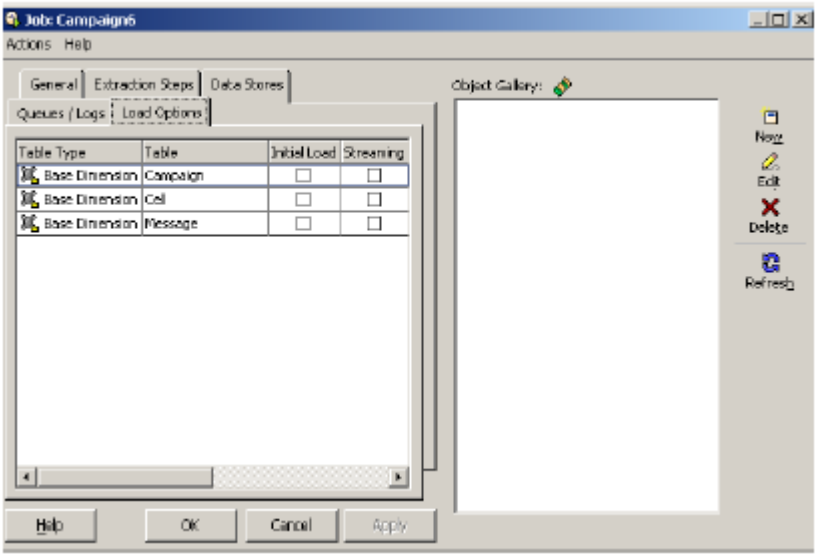


Figure 99: Job Dialog Box: Load Options Tab

- Select **Initial Load** if the table must be reloaded from its staging table, without regard to the current contents of the dimension table. When you check this option, you must also assign an Initial Load semantic to any extraction step that populates this table. (See "Semantics," on page 223 for more information about Initial Load semantics for facts and dimensions.)
- Extraction filters are disabled for any table marked as **Initial Load**. When the **Initial Load** option is checked, EpiChannel retrieves all rows in the source database that match a query, ignoring extraction filters. See "Extraction Macros" on page 480, and "How EpiChannel Identifies Data To Be Extracted" on page 362, for more information on extraction filters.

Note: When you populate a table marked as Initial Load in a job, the behavior of truncation steps is modified. Any truncation step that truncates the staging table for the fact or dimension being populated also truncates the inactive base table when Initial Load is selected for the table

- Select **Streaming** if the job must load data from the specified table directly into your data mart base tables, rather than first loading data into a staging table and then applying a semantic. When you check this option, you must also assign a Streaming semantic to any extraction step that populates this table. For dimension tables, this check box is only enabled when **Initial Load** is checked. For fact tables, if **Streaming** is checked and **Initial Load** is not checked, data is loaded directly into the fact Insert table for use by the **Transactional Incremental Streaming** semantic.

The Queues/Logs Tab

The **Queues/Logs** tab of the Job dialog box (see "Figure 100: Job Dialog Box: Queues/Logs Tab" on page 311) allows you to assign a job to an extraction queue and to monitor the progress of a running job.

The Queues Pane

The Queues pane shows the queues to which the current job is assigned. The name of the each queue is listed with its schedule type (a description of how often the job is executed within its queue).

Figure 10-5: Job Dialog Box: Queues/Logs Tab

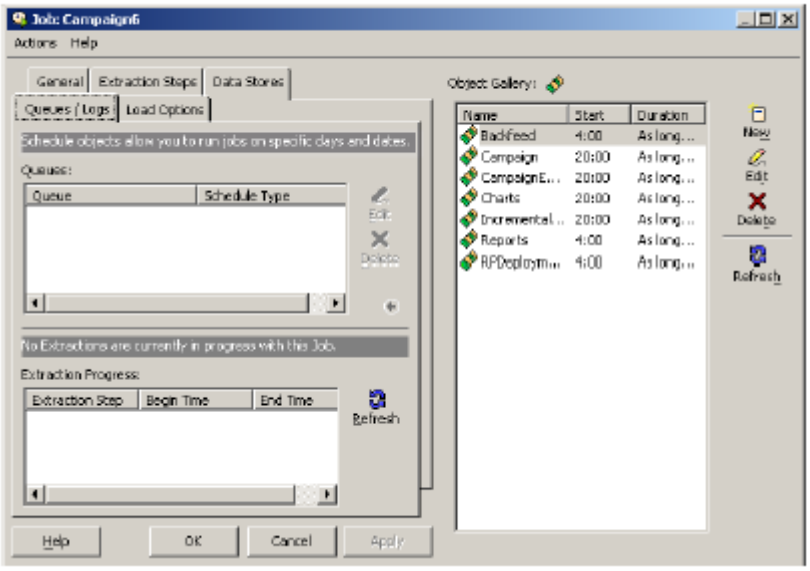


Figure 100: Job Dialog Box: Queues/Logs Tab

To schedule a job in a queue, drag the desired queue from the **Object Gallery** to the Queues pane. When you drag a queue to the Queues pane, the Task Schedule dialog box (see also, "The Queues/Logs Tab" on page 310) is displayed. Use this dialog box to specify the priority, date, time and recurrence schedule for the job. See "The Scheduler" on page 335 , for a discussion of queues and scheduling.

You can use the **Queues** subfolder in the **Extraction** folder of your EpiCenter to define a new queue. For instructions, see "Setting Up a Queue" on page 340. You can also define a new queue by clicking **New** in the **Object Gallery** .

The Extraction Progress Pane

The Extraction Progress pane displays the status of queues that are currently executing the specified job. For each extraction in progress, this is listed:

- Extraction Step : A step within the job that began executing
- Begin Time : The time at which the job step began executing
- End Time : The time at which the job step finished executing

Executing a Job

You can use the Execute Job dialog box to invoke EpiChannel with the specified job. Using the **Execute** button is equivalent to invoking the epichannel command with this job as the job option. Parameters such as database name and password, are automatically supplied. For more information about EpiChannel, see "Running Jobs with EpiChannel" on page 351

Figure 10-6: Execute Job Dialog Box

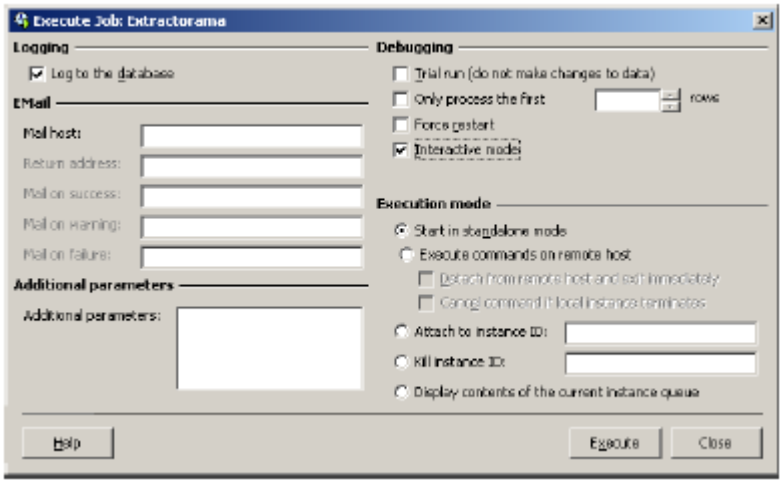


Figure 101: Execute Job Dialog Box

Note: During extraction from DB2 databases, you can encounter a fatal exception if you abort a running job and rerun it.

If an error occurs, drop the table that is being processed from the database and run Generate Schema to recreate the table.

The table below details the options available in this dialog box.

Table 61: Execute Job Dialog Box Options

Option	Description
Log to the database	Select this option if you wish to store logfiles in EpiOp. Infor recommends that you check this option for all extraction jobs.
Mail host	The name of a mail server from which EpiChannel can send mail.
Return Address	The email address that must be used for replies to EpiChannel emails
Mail on success	The email address to which notification must be sent upon successful completion of the job. You can use multiple email addresses by entering a comma-separated or semicolon-separated list

Option	Description
	of email addresses. When entering multiple addresses, do not use spaces between addresses.
Mail on warning	The email address to which notification must be sent if warnings are generated during job execution. You can use multiple email addresses by entering a comma-separated or semicolon-separated list of email addresses. When entering multiple addresses, do not use spaces between addresses.
Mail on failure	The email address to which notification must be sent if the job fails. You can use multiple email addresses by entering a comma-separated or semicolon-separated list of email addresses. When entering multiple addresses, do not use spaces between addresses.
Additional Parameters	Additional EpiChannel command line parameters that are not otherwise available in the Execute Job dialog box. See "..." on page 351 for a list of all command line parameters.
Trial run (do not change the data)	Check this option to execute the job without committing changes to metadata.
Only process the first N rows	Check this option to halt job execution after the job has processed the specified number of data rows. The value you enter is multiplied by the value of the configuration key channel_buffer_size to determine the total number of rows that are processed for each table.
Force restart	Check this option to restart the job from the beginning, regardless of any extraction history.
Interactive mode	Check this option to view user prompts and display output while the job is executing.
Start in standalone mode	Select this option to execute this job on the local machine, rather than on a remote server.
Execute commands on remote host	Select this option to execute this job on a remote server. • Choose Detach from server and exit immediately to run the job without viewing the remote EpiChannel logs as they are printed.

Option	Description
	• Choose Cancel command if client terminates to stop job execution if the remote server becomes unavailable.
Attach to instance ID	Attaches the job to another instance that is currently running. Enter the ID of the server instance to which you wish to connect in the text box.
Kill instance ID	Terminates the specified instance. Enter the ID of the server instance which you wish to kill in the text box.
Display contents of current instance queue	Displays all of the currently running instances.

Rebuilding Tables After Schema Generation

Whenever you change the structure of a fact or dimension table in your EpiCenter, you must run the Generate Schema command (see "Generating the Schema" on page 161) in order to propagate those changes from your EpiMeta database to your EpiMart database. When you generate the schema, all information about the accelerators generated by Aggbuilder (see "AggBuild" on page 298) and MomentumBuilder (see "MomBuild" on page 299) is deleted for tables that are rebuilt. You must run extraction jobs that create these tables and indexes after you generate the schema.

Note: If your Infor Campaign Management installation makes use of lists or campaigns, you must ensure that the required tables and indexes are rebuilt after schema generation. These tables and indexes must be rebuilt before you attempt to start your Infor Campaign Management Server.

Note: Accelerator information is not deleted for fact or dimension tables that are adapted, but not rebuilt, by schema generation.

To ensure that all necessary aggregates and indexes are built after you generate the schema, you must run an extraction job that applies a semantic to every table that is changed by schema generation and that includes the **AggBuild** and **MomBuild** extraction steps.

Infor recommends that, after schema generation, you run a single job that truncates all staging tables, applies semantics to all fact and dimension tables, including the built-in tables used by Infor Campaign Management (see "Star-Schema Extensions for Infor Campaign Management" on page 54), and then runs the AggBuild and MomBuild extraction steps.

In particular, after generating the schema, do not run the **Campaign** extraction job before you run a job that makes the necessary updates for the list-producing dimension tables. The **Campaign** job makes updates to the **Communication** and **Message** fact tables for each list-producing dimension. If the **Communication** and **Message** tables are not updated in the same job as the demographic tables, you must ensure that the job that updates the demographic tables runs before the job that updates the **Communication** and **Message** tables.

After rebuilding any part of the schema, when you run the next extraction job to repopulate the table, you must use an initial load semantic.

Required Semantics for Changed Dimensions

In order to ensure that aggregates are properly maintained, every job that includes a dimension semantic that changes history (that is, a dimension semantic that changes the values in base dimension rows) must also include semantics for all fact tables that join to that base dimension. All semantic types that have the **Base Updates** option set to **Any Update** or **Secondary Update**, or that have the **New Rows** option set to **Dimension Streaming** or **Stage Initial** change history.

The following pre-defined dimension semantics change the values in base-dimension rows:

- Latest Dimension Value
- Latest Dimension Value, Ignore Unknowns
- Latest Dimension Value with Fusion/Fission
- Latest Dimension Value, Preserve Fusion
- Latest Dimension Value, Rewrite All
- Slowly Changing Primary, Latest Secondary
- Initial Load Dimension
- Initial Load Dimension, Truncate Current
- Streaming Dimension

Whenever an extraction job applies one of these semantics to a base dimension, that job must also include a semantic for any fact table that joins to such a base dimension, unless no aggregates are defined on the fact table. A fact table joins to a base dimension if one of the dimension roles in the dimensionality of the fact table refers to that base dimension.

Note: If you do not apply a semantic to a fact table that joins to a base dimension table with changed values, then the correct aggregates are not built on that fact table when you execute the AggBuild extraction step. This results in incorrect answers to end-user queries.

Job Consistency

Before first executing an extraction job, you must verify that it is configured correctly. Incorrect job configuration can result in inconsistent data in your EpiCenter. By default, EpiChannel runs the Scrutiny validation utility to verify job configuration. If Scrutiny identifies a misconfigured job, EpiChannel halts without making changes to your EpiCenter, and displays the relevant job validation error. For a complete list of job validation errors, see "Job Validation Checks" on page 539 .

You must run Scrutiny (see "The Scrutiny Debugging Tool" on page 386) after making any changes to an extraction job. You must also verify that your job satisfies the following requirements:

- If an extraction command is configured for streaming in a job (see "The Load Options Tab" on page 309), then a streaming semantic instance must be applied to the target fact or dimension table in the same job. No other semantics can be applied to the target table in that job.
- All streamed extraction commands for a fact or dimension must precede the streaming semantic for that fact or dimension.
- Streamed extraction commands for a fact or dimension table must be preceded by a truncation step for that table.
- A streaming semantic can only be used with streamed facts or dimensions.
- Only one streaming semantic can be applied to a fact or dimension table in a job.
- If a base dimension is mapped (that is, it does not have the **This Dimension is Unmapped** option checked) and the dimension is being loaded with an Initial Load semantic type, then all facts that include that base dimension in their dimensionality must also be loaded with an Initial Load semantic in the same job. When validating jobs, all built-in semantics with names that start with “Initial Load” or “Streaming” (with the exception of “Streaming Transactional Fact”) are considered to be Initial Load semantics.
- If an Initial Load semantic instance is applied to a fact or dimension table in a job, then the dimension or fact must be marked as **Initial Load**.
- If a semantic instance is applied to a dimension table that has the **Allow EpiChannel to Map Keys to SSKeys in Memory** option checked and that does not have the **This Dimension is Unmapped** option checked, then the semantic instance must be executed before any extraction commands that extract data for facts that include this dimension in their dimensionality.
- If a semantic that changes history (see "Required Semantics for Changed Dimensions" on page 315) is applied to a dimension, then semantics must be applied to all fact that include the dimension in their dimensionality.
- If a dimension or fact is marked as **Initial Load** , then a single Initial Load semantic must be applied to the fact or dimension table that is populated by the extraction command. This semantic must be applied after the extraction commands are executed.
- If a job includes an extraction command that populates a base dimension with integer mapped columns, then a semantic instance must be applied to that dimension before a Commit extraction step is executed.
- If a Count Unjoined semantic is applied to a fact table, then a different non-streaming fact semantic must be applied to the fact table in the same extraction job.
- If job step consists of an extraction command that populates a fact or dimension staging table, then that job step must be configured to use the EpiMart data store as the output data store. See "The Data Stores Tab" on page 308.
- All extraction commands that populate a fact or dimension table must be executed before the semantics for that table. Extraction commands that do not populate a table can be executed at any time.
- If your job includes an Obtain Campaign Backfeed Lock step, which obtains a lock on the campaign export tables, then it must also include a Release Export Table Lock step, to release the lock. No other job that writes to the backfeed tables (such as a campaign export or a backfeed job) can proceed while a lock is in place.
- In an extraction job, if you run a semantic on a dimension that has integer mapped columns, you must also run an extraction command on that dimension in the job. This is because the extraction command step is responsible for ensuring that the integer mapping tables are up to date. Running

a semantic without a corresponding extraction command on an integer mapped dimension can result in data corruption. If there is no actual data to be extracted, then you can configure a dummy extraction command, (such as

```
SELECT column list FROM source tables WHERE 0=1
```

).

Note: The use of a Commit job step (either the Commit step or the Commit Backfeed step) resets job validation.

Multiple Commit steps break a job into blocks, each of which must follow the validation rules in this section. The validation rules apply to all job steps between the beginning of the job and the first Commit step, to all job steps between two successive Commit steps, and to all job steps between the last Commit step and the end of the job.

If the only Commit step in a job is in the End of Extraction group at the end of the job, then the job validation rules apply to the entire job.

Toggling Data Mart Tables

You can use the Job Control dialog box (see "Figure 102: The Job Control Dialog Box" on page 317) to roll back, cancel and commit jobs. To display the Job Control dialog box, click the “Job Control” button on Admin Manager’s tool bar.

Figure 10-7: The Job Control Dialog Box

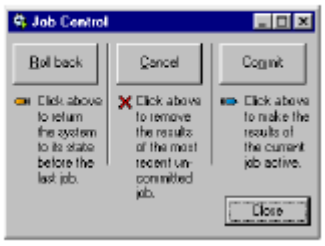


Figure 102: The Job Control Dialog Box

To switch to the set of mirrored tables that was active before the most recent extraction, click **Roll Back**. When you click this button, the state of the A/B tables for all facts and dimensions is toggled back to the previous state of these tables. The state of the backfeed tables is not changed. To switch back to the set of mirrored tables that was made active by the most recent extraction, click **Commit**. To cancel a job, roll back the job and then click **Cancel**. You cannot commit a job again after you have cancelled it.

Queries run against the set of tables that are enabled (the Current tables), and extractions populate the other set of tables. See "Data Mart Mirroring: A and B Tables" on page 191 for more information on mirrored data mart tables.

Extraction Job Templates

11

A newly initialized EpiCenter includes default extraction job templates that you can use to configure extraction jobs in your own system. Read this chapter to familiarize yourself with the structure of these jobs and the ways in which they must be configured for your EpiCenter.

Note: Do not copy and paste scripts, commands or code from this document. Line breaks and some other characters picked up when you copy from the PDF can cause errors in the pasted text.

Default Job Templates

An EpiCenter has the following default job templates: `Default`, `Campaign`, `FactRollOff`, `Backfeed`, and `BackfeedRollOff`. These job templates exist in the **Jobs > Templates** subfolder of the Job dialog box **Object Gallery** and are uneditable.

If you wish to use a template to create a new extraction job, specify a copy of both the job and the extraction steps contained within the job. You can edit the copies that you create.

The default job templates are described in the following sections.

The Default Job

The **Default** job is designed for extraction of data from your source systems.

This job includes the following steps:

- Begin Extraction , which includes:
  - Cancel In-Progress Job: An SQL extraction step that cancels any extraction jobs started in the EpiCenter but have not run to completion. This step does not cancel the Backfeed job if it is currently running.
- Dimension Extraction/Semantics: An extraction group that is meant to contain extraction commands and semantics for all dimensions in your EpiMart. This group contains the **Sample List Producing Dimension Extraction** group template, which contains an unbound truncation step and semantic instance step.

    - Duplicate this group for each dimension in your data mart (including non-list-producing dimensions) and bind and enable these steps.
    - Add the appropriate extraction step for the dimension between the truncation and semantics step (order must be truncate-extract-semantics) before executing this job.

- Fact Extraction/Semantics : A group to which you can add extraction commands and semantics for facts. This group contains the **Sample Seed Fact** extraction group template, which contains an unbound truncation step and semantics instance step.
    - Duplicate this group for each fact in your data mart (including facts derived from list-producing dimensions as well as regular fact tables) and bind and enable these steps.
    - Add the appropriate extraction step for the fact between the truncation and semantics step (order must be truncate-extract-semantics) before executing this job.

- Builders : A group that contains extraction steps that build accelerators for your EpiCenter.
    - `AggBuild` : An extraction step that invokes the Agg Builder utility, which builds aggregates in your data mart.
    - `MomBuild` : A extraction step that invokes the Momentum Builder utility, which builds special accelerators for lists and campaigns in your data mart.

- MaxDateMasterReplace : A group that contains extraction steps for repopulating the `last_extract_date` external table master row. This master row tracks the last extract date at the job execution date and time level, with no reference to data mart tables.

  If you wish to distinguish the last extract dates for different fact and dimension tables, disable this group and use the `MaxDateReplaceAll` extraction group instead (available in **Extraction Steps for Customization > MaxDate**). This group includes a single truncation step, but is intended to be customized with multiple steps to insert rows for each extracted fact and dimension table. See "End of Extraction" on page 174 for information on populating `MaxDate` .

- End of Extraction , which includes:
    - End of Extraction (Mandatory Steps) . A group that contains the following steps, which must be executed at the end of every extraction job:
        - Max of MaxDate : An extraction command that finds the highest value extracted by `MaxDate` and records it as the last extract date. See "End of Extraction" on page 174 for information on populating `MaxDate` and configuring this extraction command.
        - Commit Job : A step that toggles the mirrored A/B tables for fact and dimension tables changed by the job.
        - Create Current Views : A step that creates views of data mart tables for external reporting.

- Refresh : An extraction step that directs the Infor Campaign Management Server to reload information from the EpiMeta database. After reloading this metadata, the Infor Campaign Management Server directs all queries against the newly extracted data in the EpiMart database. See "Refresh" on page 300 for information on the parameters that must be configured for this step.

The Backfeed Job

The Backfeed job is designed to update the main backfeed tables with data from recently run campaigns. It inserts data from the P/Q backfeed tables into the A/B backfeed tables (see "Mirroring in Backfeed Tables" on page 191, for a discussion of these tables). This job does not specified any changes to the main EpiMart fact and dimension tables. The Backfeed job consists of the following steps:

- Begin Campaign Backfeed , which consists of the following:
  - Obtain Campaign Backfeed Lock: Locks the campaign export tables so that no other job can modify them while this one is running. If the backfeed lock is already held by another job (such as a campaign export) then this step waits until the lock becomes available.
  - Cancel In-Progress Backfeed Jobs : An extraction command that cancels any backfeed jobs that are started in the EpiCenter but have not run to completion.
- Campaign Backfeed Dimension Semantics: A group containing a semantic that updates the dimension backfeed tables. The semantic merges data from the P/Q backfeed dimension tables into the A/B dimension tables. You do not need to modify this semantic for the job to execute properly.
- List-Producing Fact Backfeed Semantics: This group provides you with two options for running fact backfeed semantics:
  - The Fact Backfeed Semantics: All Backfeed Steps semantic updates all **Communication** and **Message** backfeed fact tables for all list-producing dimensions. The semantic merges data from the P/Q backfeed fact tables into the A/B fact tables. This step is enabled by default and does not require any modifications.
  - If you wish to run semantics and toggle states on a subset of your backfeed fact tables (for example, because you do not use messages in your campaigns and do not wish to process the **Message** facts) you can use the **Sample Backfeed Fact Semantics** group template:
    - Disable the Fact Backfeed Semantics: All Backfeed Steps.
    - Create a copy of the Sample Backfeed Fact Semantics group template for each list-producing dimension in your data mart.
    - Bind job steps in the group template to the communication or message fact tables that you wish to process.
    - Enable the steps.
- End of Campaign Backfeed , which consists of the following:
  - Commit Backfeed Job: A step that toggles the A/B, and P/Q backfeed tables.
  - Create Current Views: A step that creates views of data mart tables for external reporting.
- Release Export Table Lock , which consists of the following:
  - Release Backfeed Lock: Releases the lock on the backfeed tables which is obtained during the first step of this job. Each job must contain as many steps for releasing the backfeed lock as for establishing it.

Infor recommends that you use an extraction job based on this job template, with its associated queue, to update your backfeed tables without extracting backfeed data to the EpiMart database.

The Campaign Extraction Job

The **Campaign** default job is designed for extraction of campaign data from the EpiOp backfeed tables into the EpiMart. The structure of this job is similar to the structure of the default source-system extraction jobs, but it also includes steps to update backfeed data before extracting that data to the EpiMart database. This job consists of the following steps:

- Begin Campaign Backfeed , Campaign Backfeed Dimension Semantics , List Producing Fact Backfeed Semantics , and End of Campaign Backfeed groups. These groups are the same as those in the Backfeed job template (see "The Backfeed Job" on page 321) and are configured in the same way.
- Begin Extraction
- Campaign Dimension Extraction : A group with SQL extraction steps and semantics that copy data from the backfeed tables into your `Campaign` , `Cell` , and `Message` dimensions. You do not need to modify these extraction steps unless you have changed the structure of one of these tables. This group contains the following subgroups:
  - Campaign: This group contains a truncation step for the Campaign staging table and an extraction command that copies new data from the Campaign backfeed table into the Campaign staging table.
  - Cell: This group contains a truncation step for the Cell staging table and an extraction command that copies new data from the Cell backfeed table into the Cell staging table.
  - Message: This group contains a truncation step for the Message staging table and an extraction command that copies new data from the Message backfeed table into the Message staging table.
  - Dimension Semantics : This group contains semantic instances that merge data into the Campaign , Cell , and Message dimension tables from the associated staging tables.
- List-Producing Fact Backfeed-to-Mart Extraction: A group that is meant to contain extraction commands for all list-producing-dimension fact tables. This group initially contains the Sample Backfeed Fact Extraction group template, which you must duplicate for every list-producing dimension in your data mart.

  The Sample Backfeed Fact Extraction group template contains the following unbound, disabled steps.
  - Communication Fact: This group contains a truncation step for the Communication staging table and an extraction command that copies new campaign data from the Communication backfeed table into the Communication staging table for the dimension.
  - Message Fact: This group contains a truncation step for the Message staging table and an extraction command that copies new campaign data from the Message backfeed table into the Message staging table for the dimension.
  - Inferred Response Fact: This group contains a truncation step for the InferredResponse staging table and an extraction command that copies new inferred response data from the Communication backfeed table into the Communication staging table for the dimension.
  - Fact Semantics: This group contains semantic instances that merge data into the `Communication` , `Message` , and `InferredResponse` fact tables from the associated staging tables.

  The job steps that are included in the **Sample Backfeed Fact Extraction** group template are shipped without being bound to any fact tables. You must bind these extraction steps to the correct backfeed fact tables and enable the job steps for the job to execute properly.

Note: The Sample Backfeed Fact Extraction group template assumes that the list-producing dimension that corresponds to the fact group is enabled for inferred response (IFR). If this is not true, you do not need to bind and enable the IFR extraction steps.

- Campaign Undo Extraction: This group copies the `campaign_undo` table from the EpiOp database to your EpiMart. This table is used to update your data mart tables when a campaign that has already been extracted from backfeed tables is undone. See "Campaign Undo" on page 201 for more information about undoing campaigns.
- List-Producing Fact Undo Campaigns: This group contains an extraction group template named Sample Undo Campaign Fact Entries that is meant to be duplicated for each list-producing dimension in your data mart. This template group contains semantic instances that debook all rows in `Communication` , `Message`, and `Inferred Response` fact tables that refer to campaigns that are undone.

  For each list-producing dimension you must bind each semantic to the appropriate fact table and enable the extraction command.

  See "Campaign Undo" on page 201 for more information about undoing campaigns.

  Note: The Sample Undo Campaign Fact Entries group template assumes that the list-producing dimension that corresponds to the fact group is enabled for inferred response (IFR). If this is not true, you do not need to bind and enable the IFR semantic.

- Builders
- MaxDateMasterReplace
- End of Extraction

Infor recommends that you use an extraction job based on this job template, with its associated queue, to extract data from the backfeed tables.

Note: After you generate your schema, you must run a job that builds the appropriate Aggregate- and Campaign-related tables and indexes before running the Campaign job. The requirements for this job are discussed in "The Campaign Extraction Job" on page 322.

Note: Campaign export and campaign extraction both operate on the backfeed tables. When one of these jobs starts, it obtains a lock on the backfeed tables. If another export or backfeed job starts before the first is finished, it must wait until the backfeed lock are released. The Campaign, backfeed, and backfeed rolloff jobs include job steps for obtaining and releasing the backfeed lock.

The Fact Roll-Off Job

If you wish to roll off historical transactional fact table data that is no longer needed, you can run a roll-off job. Fact roll-off is performed by the special **Rolloff** job step. When you run a job that uses the **Rolloff** job step, historical data in all fact tables that are configured for roll-off is deleted. Only data that is older than the specified roll-off period is deleted.

The **FactRolloff** job is designed to delete old fact data from fact tables that are configured for roll-off (see "Fact Roll-Off" on page 176, and "Fact Tables" on page 120).

Fact roll-off must only be performed when all mirrored tables contain the same data. The **FactRolloff** job is configured to ensure that this condition is satisfied. You must add the appropriate truncation steps and semantic steps to this job before running it.

The **FactRolloff** job consists of the following steps:

- Begin Extraction
- MakeFactsEqual : An extraction group that is designed to ensure that the mirrored A and B fact tables contain the same data. You must add extraction steps to the first two steps in this group. This group consists of the following steps:
  - Truncate Fact Stage Tables: An empty group to which you must add truncation steps for all fact staging tables for facts that are to be rolled off.
  - Incremental Fact Semantics: An empty group to which you must add Transactional semantics for all fact tables that are to be rolled off. These semantics, when run with empty staging tables, ensure that the inactive mirrored tables are populated with the same data that is in the Current tables.
  - Builders
- **Rolloff Checkpoint** : An extraction group that commits the previous actions and toggles the mirrored EpiMart tables. This group contains the following steps:
  - Commit Job
  - Create Current Views
- **Fact Rolloff**: This step rolls off data on the inactive mirrored copies of all facts that are configured for roll-off.
- **Rolloff Checkpoint**: This group is repeated to toggle the mirrored EpiMart tables, so that roll-off can be performed on the other mirrored copies of the fact tables.
- **Fact Rolloff**: This step rolls off data on the inactive mirrored copies of all facts that are configured for roll-off. After this step has completed, both mirrored copies of all fact tables are rolled off.
- **MaxDateMasterReplace**
- **End of Extraction**

Infor recommends that you use an extraction job based on this job template to perform fact roll-off.

The Backfeed Roll-Off Job

You can choose to roll off backfeed data that is older than a specified period of time. You use the special **Backfeed Rolloff** semantic to roll off backfeed data that is older than the time period specified in the $$MAX_BF_ROWS_AGE_IN_DAYS macro.

A backfeed roll-off job must first empty the backfeed export (P and Q) tables and then synchronize the main (A and B) backfeed tables. When the A and B tables are identical and contain all backfeed data, the job must then apply the **Backfeed Rolloff** semantic to the Communication and Message backfeed tables.

The **BackfeedRolloff** job consists of the following steps:

- Begin Campaign Backfeed: As for the Backfeed job, this includes a step for obtaining a lock on the campaign export tables.
- Campaign Backfeed Dimension Semantics
- List-Producing Fact Backfeed Semantics
- Backfeed Rolloff Checkpoint: An extraction group that commits the previous actions and toggles the mirrored EpiMart tables. This group contains the following steps:
  - Commit Backfeed Job
  - Create Current Views
- Release Export Table Lock , which consists of the following:
  - Release Backfeed Lock : Releases the lock on the backfeed tables which is obtained during the first step of this job. Each job must contain as many steps for releasing the backfeed lock as for establishing it.
- Rolloff Backfeed Semantics: Backfeed Fact Rolloff: An extraction step that rolls off historical data from all backfeed fact tables on the inactive side of your EpiOp. This step rolls off all data in the backfeed tables that is older than the time period specified in the $$MAX_BF_ROWS_AGE_IN_DAYS macro.
- Backfeed Rolloff Checkpoint: This step is executed in order to switch the mirrored tables, so that the next step operates on the other set of tables.
- Rolloff Backfeed Semantics: Backfeed Fact Rolloff
- End of Campaign Backfeed

  Note: If the backfeed tables are changed by a campaign export after the Commit Backfeed step is executed, but before the Backfeed Roll-Off semantic is executed, then the Backfeed Roll-Off semantic fails. If the Backfeed Roll-Off semantic fails, then no data is rolled off from the backfeed tables, and you must run the BackfeedRolloff job again in order to roll off backfeed data.

  Note: Do not run a campaign during the backfeed rolloff job as data can be lost. If you are running a campaign, you shut down the campaign queue before you run the backfeed and backfeed rolloff jobs. You can then re-enable the campaign queue.

Campaign Dimension Extraction Steps

If you are using Infor Campaign Management, you need to extract data for the **Campaign**, **Message**, and **Cell** dimensions from the backfeed tables. Infor Campaign Management include default extraction commands for this purpose. Ordinarily, these extraction commands are run as part of the default **Campaign** job (see "The Campaign Extraction Job" on page 322).

The following sections provide customization information for these extraction steps.

Note: While Campaign, Message, and Cell dimension data are ordinarily extracted from the backfeed tables, these dimensions can also be populated from external data sources.

Campaign Dimension Extraction

The default extraction command for the **Campaign** dimension has the following SQL code:

```
SELECT

campaign_sskey,

$$TO_EPIDATE[ rundatetime ] date_modified,

$$FOREACH_SKIP[COL ~,~ $$DIMCOL ~,~

$$NVL[$$COL~.~ETLCOL~,~$$COL~.~ETLUNKNOWN] $$COL, ~,~ createdatetime
]

$$TO_EPIDATE[$$NVL[createdatetime~,~$$DBNOW]] createdatetime

FROM

$$DIMBFT$$CURRBF

WHERE $$DATE_FILTER[rundatetime]

 AND is_inference_campaign = 0
```

This command copies values from backfeed-table columns to the dimension columns with the same names. As usual, the `$$NVL` macro must be used to ensure that no null values are copied. The backfeed table has a column called `rundatetime`, which records the time that the campaign was run, that must be used for the date. BIGDATE data type is supported by the `rundatetime` column. This column must also be used for extraction subset filtering, as shown in the `WHERE` clause of the default extraction step. See "How EpiChannel Identifies Data To Be Extracted" on page 362, for a discussion of extraction subset filtering.

The Campaign backfeed table stores information for inference campaigns in addition to information for ordinary campaigns. Inference campaigns are identified by a value of `1` in the `is_inference_campaign` column. Since inference campaign information is ordinarily not stored in the **Campaign** dimension, the extraction code has a `WHERE` clause that excludes inference campaign data.

You may also want to use the `$$NBIN_*` and `$$CBIN_*` macros to divide numerical values, such as costs, into ranges. See "Base Dimension Staging SQL Statements" on page 285.

The mirrored backfeed table from which data is being extracted is identified by the macro expression `$$DIMBFT$$CURRBF`. See "SQL Macros" on page 182, and "Appendix A, “Infor Campaign Management Macros,”" on page 441 for a discussion of Infor Campaign Management macros. See "Mirroring in Backfeed Tables" on page 191, for a discussion of mirrored backfeed tables.

Cell Dimension Extraction

The default extraction command for the **Cell** dimension has the following SQL code:

```
SELECT
cell_sskey,
$$TO_EPIDATE[ $$DIMBFT$$CURRBF.rundatetime ] date_modified,
$$FOREACH_DELIM[, ~,~ COL ~,~ $$DIMCOL ~,~
$$NVL[$$COL~.~ETLQBFCOL ~,~ $$COL~.~ETLUNKNOWN] $$COL]
FROM
$$DIMBFT$$CURRBF, campaign_bf$$CURRBF c
WHERE $$DATE_FILTER[$$DIMBFT$$CURRBF.rundatetime]
 AND $$DIMBFT$$CURRBF.campaign_id = c.campaign_id
 AND c.is_inference_campaign = 0
 AND $$DIMBFT$$CURRBF.is_tracking_cell = 0
```

The same considerations apply to this extraction command as to the extraction command for the **Campaign** dimension. Inference campaign data is excluded by joining to the Campaign backfeed table.

The Cell backfeed table includes the `is_tracking_cell` column, which is used to mark rows that keep track of intermediate results for multi-wave campaign. Only rows with is_tracking_cell = 0 must be extracted.

The built-in columns of the **Campaign** , **Message,** and **Cell** dimensions are described in "Appendix D, “Campaign-Related Dimensions.”" on page 509

Message Dimension Extraction

The default extraction command for the **Message** dimension has the following SQL code:

```
SELECT
message_sskey,
```

```
$$TO_EPIDATE[ $$DIMBFT$$CURRBF.rundatetime ] date_modified,

$$FOREACH_DELIM[, ~,~ COL ~,~ $$DIMCOL ~,~

$$NVL[$$COL~.~ETLQBFCOL ~,~ $$COL~.~ETLUNKNOWN] $$COL ]

FROM

$$DIMBFT$$CURRBF, campaign_bf$$CURRBF c

WHERE $$DATE_FILTER[$$DIMBFT$$CURRBF.rundatetime]

AND $$DIMBFT$$CURRBF.campaign_id = c.campaign_id

 AND c.is_inference_campaign <> 1
```

The same considerations apply to this extraction command as to the extraction command for the **Campaign** dimension. Inference campaign data is excluded by joining to the Campaign backfeed table.

The built-in columns of the **Campaign** , **Message,** and **Cell** dimensions are described in "Appendix D, “Campaign-Related Dimensions.”" on page 509

List-Producing Fact Extraction Steps

If you are using Infor Campaign Management, you need to extract data for all **Communication** and **Message** fact tables from the backfeed tables. A **Communication** and **Message** fact table is created for each backfeed, list-producing base dimension. Your EpiCenter includes an extraction group template for this purpose in the default **Campaign** job. (See "Built-in Fact Tables" on page 55 for information on list-producing fact tables.)

Additionally, if you have configured Infor Campaign Management to allow end users to run inference campaigns, then you need to extract data into the `InferredResponse` fact table for each list-producing dimension. Inference campaigns allow end users to infer campaign responses when campaign response data is not available directly.

Communication Fact Table Extraction

The default extraction command for a **Communication** fact table has the following SQL code:

```
SELECT

f.campaign_sskey $$CAT '~~~' $$CAT $$NVL[$$LPDIMSSKEYR ~,~ 'UNKNOWN']
```

```
 ss_key,
$$TO_EPIDATE[ f.rundatetime ] date_key,
$$TRANSTYPE[BACKFEED] transtype_key,
$$CIF[$$NO_PROCESS_KEY~,~0~,~1~,~] process_key,
$$NVL[$$LPDIMSSKEYR ~,~ 'UNKNOWN'] $$LPDIMSSKEYR,
f.campaign_sskey,
f.cell_sskey,
$$NVL[backfeedseq_key~,~0] backfeedseq_key,
1 occur
FROM
$$FCTBFT$$CURRBF f, campaign_bf$$CURRBF[Campaign] c,
cell_bf$$CURRBF[Cell] cl
WHERE $$DATE_FILTER[f.rundatetime]
AND f.campaign_sskey = c.campaign_id
AND c.is_inference_campaign = 0
AND f.cell_sskey = cl.cell_sskey
AND cl.is_tracking_cell = 0
```

As with the extractors for the Campaign, Message, and Cell dimensions, most columns are simply copied from the backfeed table. Note the following:

- $$LPDIMSSKEYR represents the sskey of the list-producing dimension that corresponds to the communication fact table.
- The communication backfeed table has a column called `rundatetime` , which records the end time of the campaign. (More specifically, it records the update time for the backfeed tables). This column must also be used for extraction subset filtering, as shown in the `WHERE` clause of the default extraction step. See "How EpiChannel Identifies Data To Be Extracted" on page 362, for a discussion of extraction subset filtering.
- The extraction code is designed to work with a list-producing dimension that has a string `sskey` value. If the list-producing dimension for the Communication table uses integer `sskey` s, you must replace the string `'UNKNOWN'` in the expression for the `sskey` with an appropriate integer value.

For example, use the following line in the extraction code for the $$LPDIMSSKEYR column if your dimension uses integer `sskey` s:

```
$$NVL[$$LPDIMSSKEYR ~,~ 1] $$LPDIMSSKEYR,
```

- The default `transtype` value for a communication, which indicates that a treatment is applied, and specified using the `$$TRANSTYPE[BACKFEED]` macro. Additional values can be defined for other campaign-related facts.
- The **occur** column indicates that the fact has occurred. It must be given a value of `1` .
- The backfeed table from which data is being extracted is identified by the macro expression `$$FCTBFT$$CURRBF` . See "SQL Macros" on page 182, and "Appendix A, “Infor Campaign Management Macros,”" on page 441 for a discussion of Infor Campaign Management macros.
- Communication backfeed tables include data for inference campaigns, which is not normally extracted to the **Communication** fact table. Inference campaign data is excluded by joining to the Campaign backfeed table and excluding all Communication backfeed rows that join to a Campaign backfeed row with a value of `1` in the `is_inference_campaign` column.
- Cell backfeed tables include data for incomplete multi-wave campaigns. This data is not normally extracted to the **Communication** fact table, and it is excluded by joining to the Cell backfeed table and excluding all Communication backfeed rows that join to a Cell backfeed row with a value of 1 in the `is_tracking_cell` column.

Message Fact Table Extraction

The default extraction command for a **Message** fact table has the following SQL code:

```
SELECT

f.campaign_sskey $$CAT '~~~' $$CAT $$NVL[$$LPDIMSSKEYR ~,~'UNKNOWN']
 ss_key,

$$TO_EPIDATE[ f.rundatetime ] date_key,

$$TRANSTYPE[BACKFEED] transtype_key,

$$CIF[$$NO_PROCESS_KEY~,~0~,~1~,~] process_key,

$$NVL[$$LPDIMSSKEYR ~,~ 'UNKNOWN'] $$LPDIMSSKEYR,

f.campaign_sskey,

f.cell_sskey,
```

```
f.message_sskey,
$$NVL[placement_key~,~'UNKNOWN'] placement_key,
$$NVL[backfeedseq_key~,~0] backfeedseq_key,
1 occur
FROM
$$FCTBFT$$CURRBF f, campaign_bf$$CURRBF[Campaign] c,
cell_bf$$CURRBF[Cell] cl
WHERE $$DATE_FILTER[f.rundatetime]
AND f.campaign_sskey = c.campaign_id
 AND c.is_inference_campaign = 0
 AND f.cell_sskey = cl.cell_sskey
 AND cl.is_tracking_cell = 0
```

Refer to the notes for the **Communication** fact table for additional information about **Message** fact table extraction.

InferredResponse Fact Table Extraction

If you have configured Infor Campaign Management to allow end users to run inference campaigns, then you need to extract data into the `InferredResponse` fact table. Inference campaign data is stored in the backfeed tables along with ordinary campaign data.

The default extraction command for an `InferredResponse` fact table has the following SQL code:

```
SELECT
$$NVL[comm.cell_sskey ~,~ 'UNKNOWN'] $$CAT ':' $$CAT
$$NVL[comm.$$LPDIMSSKEYR ~,~ 'UNKNOWN'] ss_key,
$$TO_EPIDATE[ ca_bf.rundatetime ] date_key,
$$TRANSTYPE[CAMPAIGN_RESPONSE] transtype_key,
```

```
$$CIF[$$NO_PROCESS_KEY~,~0~,~1~,~] process_key,

$$NVL[ comm.campaign_sskey ~,~ 'UNKNOWN' ] campaign_sskey,

$$NVL[ comm.cell_sskey ~,~ 'UNKNOWN' ] cell_sskey,

$$NVL[ comm_bf.$$LPDIMSSKEYR ~,~ 'UNKNOWN' ] $$LPDIMSSKEYR,

$$TO_EPIDATE[ ca.rundatetime ] campaigndate_key,

1 Occur

FROM

Campaign_BF$$CURRBF[Campaign] ca_bf,

$$LPDIMROOT[]_Com_BF$$CURRBF[$$LPDIMROOT[]_Com] comm_bf,

$$LPDIMROOT[]_Com_BF$$CURRBF[$$LPDIMROOT[]_Com] comm,

Campaign_BF$$CURRBF[Campaign] ca,

Cell_BF$$CURRBF[Cell] cl

WHERE $$DATE_FILTER[ca_bf.rundatetime] AND

comm.cell_sskey = cl.cell_sskey AND

cl.is_tracking_cell = 0 AND

ca_bf.campaign_sskey = comm_bf.campaign_sskey AND

ca.campaign_code = ca_bf.track_campaign_code AND

comm.campaign_sskey = ca.campaign_sskey AND

comm_bf.$$LPDIMSSKEYR = comm.$$LPDIMSSKEYR AND

ca_bf.is_inference_campaign = 1 AND

ca.is_inference_campaign = 0 AND NOT EXISTS (

  SELECT 1 FROM $$LPDIMROOT[]_Com_BF$$CURRBF[$$LPDIMROOT[]_Com~,~Fact
BF] other,

  Campaign_BF$$CURRBF[Campaign] other_ca

  WHERE other.rundatetime > comm.rundatetime AND

  other_ca.campaign_sskey = other.campaign_sskey AND

  comm.$$LPDIMSSKEYR = other.$$LPDIMSSKEYR AND
```

```
other_ca.campaign_code = ca.campaign_code AND

other_ca.is_inference_campaign = 0

)
```

This extraction code extracts data about the inference campaign and the original campaign for which responses are inferred. The `InferredResponse` table includes the special `campaigndate` dimension role, which refers to the **Date** dimension, in its dimensionality. For each inferred response, `date_key` refers to the date of the inferred response and `campaigndate_key` refers to the date of the original campaign.

The extraction code is also designed to work with a list-producing dimension that has a string `sskey` value. If the list-producing dimension for the **InferredResponse** table uses integer `sskey` s, you must replace the string `'UNKNOWN'` in the expression for the `sskey` with an appropriate integer value. For example, you can use the following line in the extraction code for the $$LPDIMSSKEYR column if your dimension uses integer `sskey` s:

```
$$NVL[ comm_bf.$$LPDIMSSKEYR ~,~ 1 ] $$LPDIMSSKEYR,
```

Note: For inference campaigns to be extracted properly from EpiOp to EpiMart, the Parent and Tracking campaign codes must be specified.

Extracting Real-Time Data

If you are using Infor Real-Time (Infor Interaction Advisor), you can use the pre-defined extraction commands that Infor provides to extract Real-Time data into your EpiCenter. These extraction commands are described in chapter 13, “Integrating Infor Campaign Management with Infor Interaction Advisor,” in the *Topic Implementation Guide*. You can customize these extraction commands to reflect details of your EpiCenter schema.

You can also define custom extraction commands to extract data from the Real-Time database (RTDB). Refer to chapter 14, “The RTDB,” in the *Real-Time Implementation Guide*, for a detailed description of the RTDB. Refer to Real-Time Applications Library online help, available from the Real-Time Studio Help menu, for a description of the e-Commerce template, which provides much of the data in the RTDB.

The Scheduler

12

In a production EpiCenter, extraction jobs are normally executed on a regular schedule. You can schedule jobs to be executed automatically with the Infor Campaign Management Scheduler.

The Scheduler is responsible for running processes that are scheduled for automatic execution by Infor Campaign Management applications. Typical Scheduler tasks include running extraction jobs and generating campaigns. Ordinarily, the Scheduler is run as an automatic-startup Windows service on the same host as the Infor Campaign Management Server. You can also run the Scheduler from a command line. (See Chapter 10, “Infor Campaign Management Server,” in the *Infor Campaign Management Topic Implementation Guide* for more details.)

When working with the Scheduler, be aware of two important points:

- Two systems cannot run a Scheduler simultaneously on the same EpiMeta database.
- Every Scheduler instance is associated with an Infor Campaign Management Server instance. The Scheduler service always uses the EpiMeta database that is used by the associated Infor Campaign Management Server. The scheduling information in an EpiMeta database is configured using Admin Manager.

Note: In order to schedule extraction jobs, you must have EpiChannel installed as a service or daemon. See the Infor Campaign Management Installation Guide for details on installing EpiChannel as a service or daemon.

Note: Oracle 11g users using 64-bit HPUX with the 64-bit Infor Campaign Management application can use the 64-bit Oracle libraries at $ORACLE_HOME/lib. All other Oracle 11g users must use the 32-bit versions of the Oracle libraries when running the Scheduler and other Infor Campaign Management executables. To use the 32-bit libraries, set LD_LIBRARY_PATH (Sun), SHLIB_PATH (HP), or LIBPATH (AIX) to include $ORACLE_HOME/lib32 instead of $ORACLE_HOME/lib.

Queues

The Scheduler assigns tasks to queues , each of which can contain several different tasks. The default Infor Campaign Management configuration has the following built-in queues: **Backfeed**, **Campaign**, **CampaignExtract**, **Charts**, **IncrementalExtract**, **Reports**, and **RPDeployments**.

Within a queue, tasks are run based on their scheduled start time and a user-assigned priority level. Multiple tasks in a queue can be run in parallel. The Scheduler can run queues in parallel, and queues are scheduled based on queue schedule configurations and queue dependencies .

The option Run Queue after successful completion of, on the Dependencies tab, allows you to specify a conditional queue kickoff option:Ignore Dependency, All Tasks, and Any Tasks.

Queue Dependencies

Queue dependencies are assigned in Admin Manager. (See "Setting Up a Queue" on page 340.) If Queue B is configured to depend on Queue A, then the two queues are never run at the same time. If both queues are ready to start at the same time, then Queue A is run before Queue B. If Queue B depends on Queue A, then Queue A can also be configured to kill (terminate) Queue B whenever necessary. If Queue A can kill Queue B, then if Queue A becomes ready to run while Queue B is running, Queue B is terminated and Queue A is started (unless Queue A depends on another queue that is ready to run).

To illustrate queue dependencies, consider the following possible relationships among four queues: the Charts queue depends on the Campaign queue, the Campaign queue depends on both extract queues, and The WeeklyExtract queue depends on the DailyExtract queue. In addition, both extract queues can kill the Campaign queue, and the Campaign queue can kill the Charts queue. These dependencies are illustrated in "Figure 103: Typical Queue Direct Dependencies" on page 336.

Note: The illustrations do not show the default dependencies in an EpiCenter. The dependencies in these figures are used for illustration only.

Figure 12-1: Typical Queue Direct Dependencies

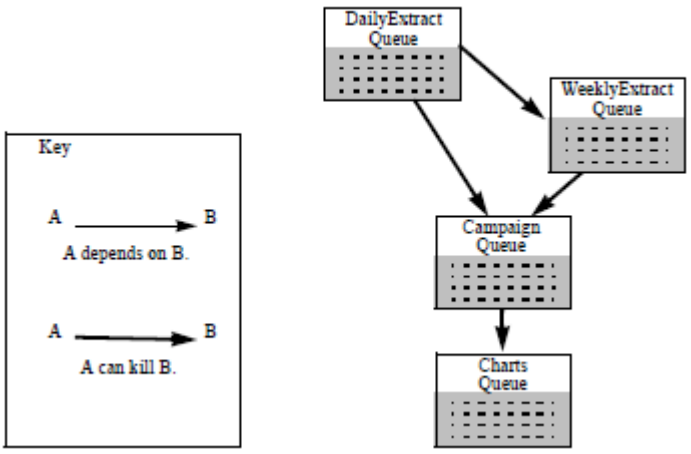


Figure 103: Typical Queue Direct Dependencies

The diagram in "Figure 103: Typical Queue Direct Dependencies" on page 336 resembles a family tree. For this reason, when Queue B depends on Queue A, Queue A is called a parent of Queue B and Queue B is called a child of Queue A.

The Infor Campaign Management Scheduler enforces transitivity . This enforcement means that if Queue C depends on Queue B and Queue B depends on Queue A, then Queue C depends on Queue A. Similarly, if Queue A can kill Queue B and Queue B can kill Queue C, then Queue A can kill queue C. Taking transitivity into account, the dependencies of the queues in "Figure 103: Typical Queue Direct Dependencies" on page 336 can be illustrated as in "Figure 104: Typical Queue Dependencies" on page 337.

Figure 12-2: Typical Queue Dependencies

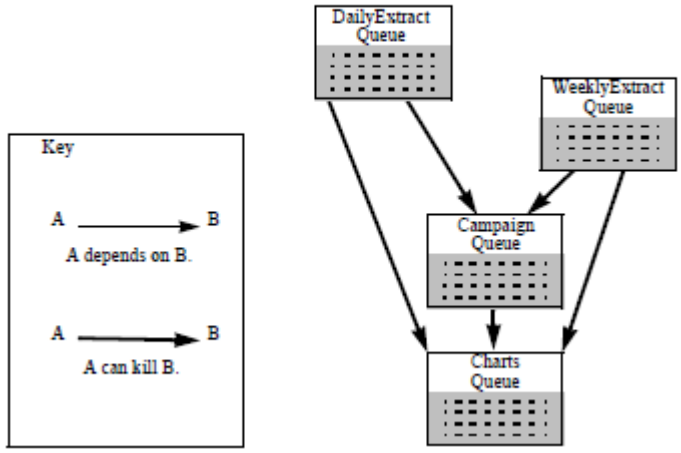


Figure 104: Typical Queue Dependencies

Since successful extraction is very important for an EpiCenter, both extraction queues are configured to be able to kill any non-extraction queue. In addition, the Campaign queue is configured to be able to kill the Charts queue, which has relatively low priority. Since two extract jobs must never be run at the same time, there is a dependency between the extraction queues. If the job in the DailyExtract queue generally completes more quickly than the job in the WeeklyExtract queue, WeeklyExtract can be configured to be dependent on DailyExtract. However, it is undesirable to have the DailyExtract queue kill the WeeklyExtract queue, since the job in the WeeklyExtract queue can be very close to completion, so the DailyExtract queue can be configured so that it cannot kill the WeeklyExtract queue.

Circular Dependencies

When configuring Scheduler queues there must not be any circular dependencies . A set of queues has circular dependencies if a chain of dependencies leads from a queue back to itself. For example, if Queue D depends on Queue B, which depends on Queue A, which depends on Queue D, then these queues are said to have circular dependencies. This is illustrated in "Figure 105: Circular Queue Dependencies" on page 338.

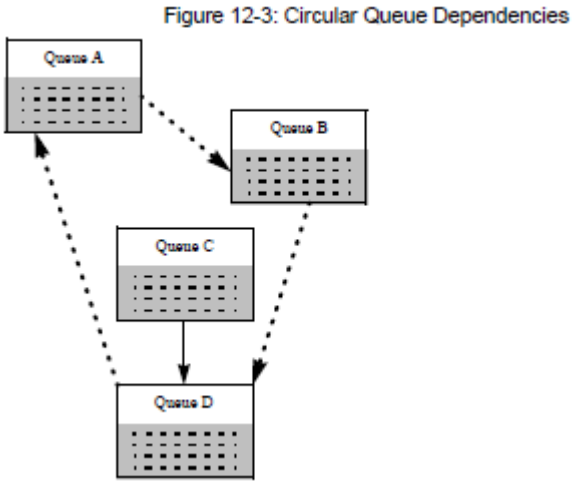


Figure 105: Circular Queue Dependencies

Note: If queues are configured to have circular dependencies, then the Scheduler does not start. No scheduled tasks are run until the circularity is removed and the Scheduler is restarted.

Scheduler Operation

Ordinarily, the Scheduler service for an EpiCenter runs at all times. Typically, the Scheduler wakes up at a regular interval, finds queues that need to be run, arranges these queues in a partial order based on their dependencies, and then runs the queues based on that order.

A queue can be in one of three states:

- Running

  The tasks in the queue are currently being executed.

- Sleeping

  The queue is not ready to run, and no tasks are being executed.

- Waiting

  The queue is ready to run, but it has a dependency on another queue that prevents it from being run. The queue runs as soon as these dependencies are resolved.

When the Scheduler starts up, it determines whether another Scheduler instance is currently running against the specified EpiMeta database. If another Scheduler instance is currently running against the EpiMeta database, then the Scheduler does not start. The Scheduler then checks for circular dependencies among queues. If circular dependencies are found, the Scheduler halts.

After it has successfully started, the Scheduler wakes up every minute. When the Scheduler wakes up, the following events occur:

- Queues that are scheduled to run are found. A queue is scheduled to run if it has a scheduled run time between the previous scheduler wakeup time and the current scheduler wakeup time.
- Queues that are ready to run are found. A queue is ready to run if it is scheduled to run and is not currently running or disabled. This includes queues that are waiting to run from a previous scheduler wake-up.
- The queues that are ready to run are arranged in a partial order based on their dependencies.
- Blocked queues in the partial order are put into a waiting state. A queue is blocked if any one of the following is true:
  - The queue is dependent on a currently running or waiting queue.
  - The queue is dependent on another queue in the partial order.
  - A currently running queue is dependent on this queue but cannot be killed by this queue.
- Every killable queue is killed. A queue is killable if it is running and it depends on a queue in the partial order that is not blocked and that can kill it.
- Every queue in the partial order that is not blocked is started.

Task Scheduling

Within a queue, tasks are run sequentially, based on a user-assigned priority level. Two kinds of tasks can be assigned to a queue:

- Jobs

  Jobs are assigned to queues individually in Admin Manager.

- End-User Tasks

  End-user tasks, such as campaigns, are assigned to queues in groups, based on the type of task (the queue role).

Tasks can be scheduled to run once or on a recurring basis. Recurring tasks can be scheduled to run daily, weekly, or monthly. A task can be configured to run some fixed number of times on its run day, or whenever the queue runs.

A task instance is a single scheduled execution of a task. So, for example, a task that is scheduled to run four times in a day has four task instances for that day. A completed task instance is a task instance that has been executed. If a task instance is not completed on its scheduled date, it can be completed on a later date. The number of days for which an uncompleted task instance must be retried can be configured for each queue. An expired task instance is a task instance that has not been executed during this retry interval. An overdue task instance is an uncompleted task instance that is not executed on its scheduled date but has not expired.

When a queue starts running, the following occurs:

- If any running queues depend on this queue, then those running queues are killed (note that the running queues must be killable if the current queue is starting).
- All overdue task instances are found and put into a list of runnable task instances.

- All tasks that are ready to run are found. A task is ready to run if it has at least one task instance scheduled for the current day that has not yet been executed.
- For each task that is ready to run, a single task instance is added to the list of runnable task instances.
- The list of runnable task instances is ordered by the assigned priority level of the tasks. If two or more task instances have the same priority level, then the task instances at that priority level are arranged based on the scheduled start date of the task instance (with earlier dates receiving higher priority).
- The Scheduler starts the first task instance in this ordering. If the queue is configured to allow parallel execution of tasks, then the Scheduler starts as many tasks as possible, up to the configured maximum number of parallel tasks.
    - When a task instance is started:
    - The expiration date of the task instance is verified. If the task instance has expired, it is not run.
    - The task instance is executed. If the task instance exceeds the per-task execution time of the queue, it is terminated.
    - If the task instance executes successfully, then it is marked as completed.
- Steps 5 on page 340 and 6 on page 340 are repeated until all task instances in the list are completed or the maximum run time of the queue is reached. If the maximum run time of the queue is reached before all tasks instances are completed, then the queue is terminated. When a queue is terminated, all running task instances in the queue are terminated.

  When the queue has finished, all waiting queues with dependencies on this queue are examined. If any of these queues are no longer blocked, then they are started.

Setting Up a Queue

In the **Extraction** folder, right-click the **Queues** icon to open the Queue dialog box ("Figure 106: Queue Dialog Box: General Tab" on page 341). This dialog box enables you to define a queue and view its contents.

1. In the **General** tab, enter the **Name** of the queue (as defined in EpiMeta), its **Developer label** (the name that is displayed within Admin Manager), and **Label** (the name that is displayed to end users).
2. Check **Enabled** in order for the queue to be run. All queues are disabled by default.
3. Indicate whether or not you want the metadata to be refreshed when the queue starts.

   Refresh metadata reloads all the metadata into the Scheduler, similar to an Infor Campaign Management Server refresh. For a job-only queue, in which metadata is loaded in a separate process, refresh metadata is an unnecessary overhead. Other queues must generally refresh the metadata before executing.

Figure 12-4: Queue Dialog Box: General Tab

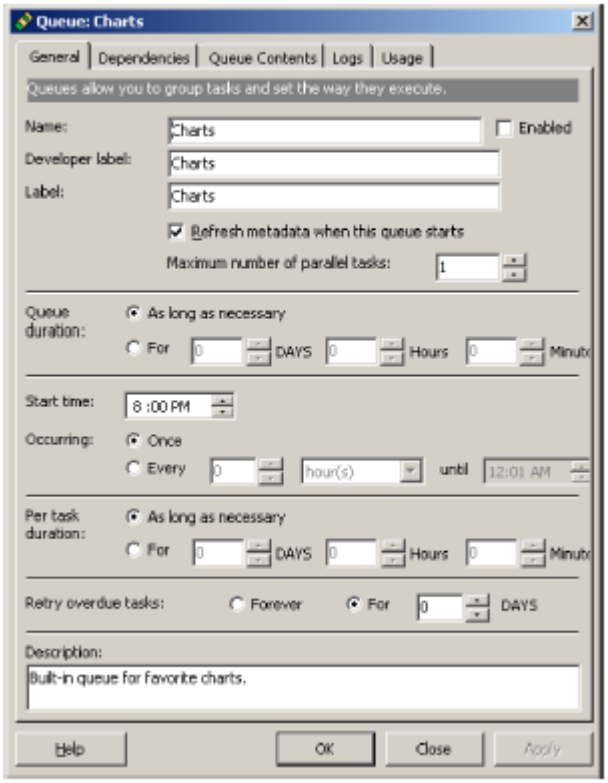


Figure 106: Queue Dialog Box: General Tab

4 Choose the **Maximum Number of Parallel Tasks** .

The scheduler attempts to run multiple tasks in the queue in parallel, up to the maximum number specified here.

Note: If you assign the Real-Time Personalization queue role to a queue, or if you do not assign any queue roles to the queue at all, the queue can only execute one task at a time.

5 Indicate a duration of the queue. To run a queue until all tasks have completed, select **As Long As Necessary** . To set a maximum running time for the queue, select **For** and specify the number of days, hours, and minutes.

If **As long as necessary** is selected, the Scheduler does not terminate the queue. If a time is indicated, the queue runs this amount of time before the Scheduler service terminates it. If a queue that can terminate the current queue is ready to run, then the current queue can be terminated even if it is configured to run as long as necessary.

6 Set the start time and indicate how often the queue must wake up: once, or every x number of hours until a specific time.

All default queues wake up once per day (8:00 p.m). The maximum number of times that a queue can wake up is once each minute of the day, or 1440 times (24 x 60 = 1440).

7 Specify the length of time that an individual task must have to complete. You can choose either **As long as necessary** , or you can specify the days, hours and minutes.

8 Indicate how long to retry tasks that are overdue: either forever, or for a set number of days.

If **Forever** is selected, then whenever a task instance is not executed for any reason, the Scheduler attempts to execute the task the next time it wakes up the queue. When **Forever** is selected, the Scheduler re-executes the task only if its original execution date is not earlier than the specified number of days.

Note: Reschedule your queues for Daylight Savings times so that they do not start during invalid HHMMSS combinations.

9 Use the **Dependencies** tab to set up dependencies between this queue and other queues expressed in parent/child relationships.

10 Normally, once a queue begins execution it does not take account of any newly scheduled tasks until it finishes execution. If you want the queue to accept new tasks and schedule them at normally scheduled times while the current queue is executing, set **Behavior > Scheduler > WakeQueueDuringExecution** to 1 using the **Configuration** dialog box (See "The Configuration Dialog Box" on page 147). The default value is 0.

11 Click **OK** to close the dialog box and save your queue.

After you have defined a queue, you can view its contents and logfiles using the other tabs in the Queue dialog box.

The Dependencies Tab

You can use the **Dependencies** tab (see "Figure 107: Queue Dialog Box: Dependencies Tab" on page 343) to set up dependencies between this queue and other queues expressed in parent/child relationships:

- Parent queues must run before child queues.
- Child queues can run only after the parent queue finishes.
- If a child queue is already running, however, and the parent queue becomes ready to run, then the parent has to wait for the child to finish (unless the parent queue is set to terminate the child queue).

When making timing and dependency selections, estimate how long your queues might run and the optimal time for them to run. Also, consider whether queues might interfere with each other.

Figure 12-5: Queue Dialog Box: Dependencies Tab

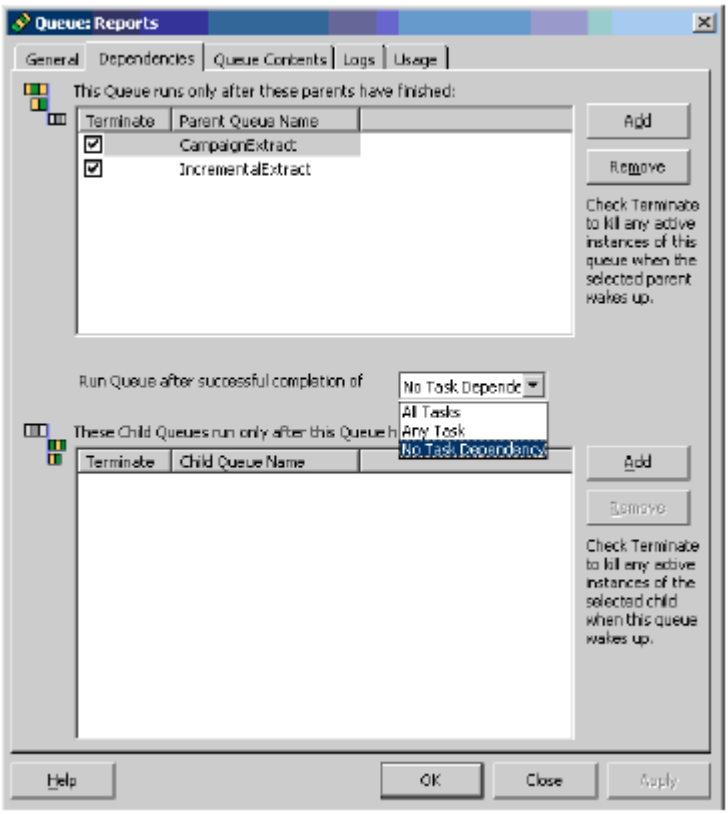


Figure 107: Queue Dialog Box: Dependencies Tab

Creating a Queue Dependency

In the upper pane of the **Dependencies** tab, click **Add** and select any parent queues of which this queue is a child.

1. Check **Terminate** to kill any instances of this queue that are running when the parent starts running.
2. In the lower pane of the **Dependencies** tab, click **Add** and select any child queues that depend on the queue you are defining as a parent queue.
3. Check **Terminate** next to a child queue, if any instance of this child queue that is running when the parent queue is ready to start must be terminated. A parent queue cannot start if a child queue is running. It has to either wait for the child queue to finish, or terminate it.
4. Click **Remove** to delete a selected dependency from the queue definition.

   Note: Before modifying any dependencies, stop the Scheduler. After setting up dependencies between queues, restart the Scheduler.

Staring a Queue using a Condition

Use the option Run Queue after successful completion of, on the Dependencies tab, to specify a condition where a queue is to be executed (also called a conditional queue kickoff). The conditions that you can specify are:

- Ignore Dependency: No condition is specified. (Default)
- All Tasks: All tasks in the parent queue completed successfully.
- Any Tasks: Any task in the parent queue completed successfully.

To create a conditional queue kick off

1. In the upper pane, click **Add** and select any parent queues of which this queue is a child.
2. Check **Terminate** to stop any instances of this queue that are running when the parent starts running.
3. On the divider bar between the upper and lower pane, select Run Queue after successful completion of, to mark the queue as executable only if the selected condition is true. Then click the drop-down button and select the condition:
   - Ignore Dependency - No condition specified. This selection is the default.
   - All Tasks - All tasks in the parent queue are completed successfully.
   - Any Tasks - Any task in the parent queue are completed successfully.
4. In the lower pane of the **Dependencies** tab, click **Add** and select any child queues that depend on the queue you are defining as a parent queue.
5. Check **Terminate** next to a child queue, to stop any instance of this child queue that is running when the parent queue is ready to start. A parent queue cannot start if a child queue is running. If the child queue is running, the parent queue must wait for the child queue to finish, or terminate the child queue.

The Queue Contents Tab

The **Queue Contents** tab displays the tasks currently in the queue, listed by name.

Figure 12-6: Queue Dialog Box: Queue Contents Tab

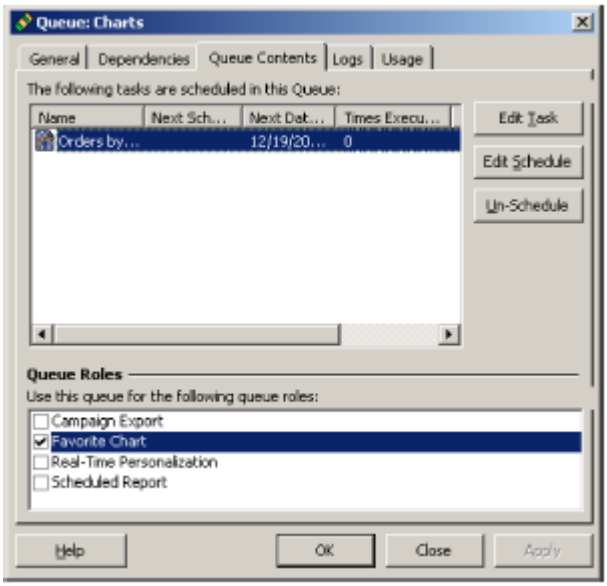


Figure 108: Queue Dialog Box: Queue Contents Tab

- The **Next Schedule Type** column shows the scheduling status of the task.
- The **Next Date (Times Executed)** column displays the next date on which this task is scheduled to be run and, in parentheses, the number of task instances scheduled for that date that have already been executed.

The **Times Executed Total** column shows the total number of times that this task has been executed.

Note: As Soon As Possible appears in the Next Schedule Type column for newly added tasks. The first time that a queue starts after a task has been added to it, As Soon As Possible is replaced by a value in the Next Date column.

Assigning Queue Roles

You can use the Use this Queue for the following Queue Roles pane to specify the types of tasks that should be assigned to your queue. Tasks are categorized by one of the following queue roles:

- Campaign Export
- Favorite Chart
- Real-Time Personalization
- Scheduled Report

Queues may be assigned multiple queue roles, or a queue might be assigned no queue roles at all (this is the case for the “Extract” queue). Each queue role may only be assigned to one queue.

Select the queue roles that you wish to assign to your queue by marking the check box next to each desired role.

Editing Tasks

You can use this dialog box to remove a task from the queue by selecting it and click **Un-Schedule** .

- To edit a scheduled task, click **Edit Task** .
- To edit the schedule or view the logs for a task, select the task and click **Edit Schedule** to open the Task Schedule dialog box (see "Figure 110: Task Schedule Dialog Box: General Tab" on page 347).

The Logs Tab

The **Logs** tab of the Queue dialog box shows logs for the queue.

Figure 12-7: Queue Dialog Box: Logs Tab

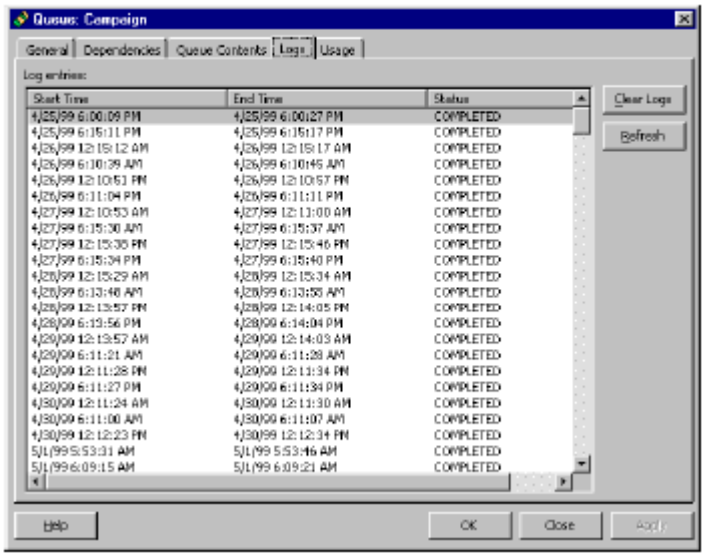


Figure 109: Queue Dialog Box: Logs Tab

A log entry contains:

- The start and end times of the queue.
- The status of the queue, such as failed, completed, or server aborted.

These logs report on whether the queue ran successfully, not whether task instances in the queue ran successfully. Therefore, a queue log can report that the queue ran successfully, although a task that ran as part of the queue failed.

Logs for individual tasks can be viewed in the Task Schedule dialog box (see "The Task Schedule Dialog Box" on page 347).

The Task Schedule Dialog Box

Use the Task Schedule dialog box to schedule tasks within queues. (A task is any job, Campaign, or Chart.) This dialog box opens when you assign a job to a queue in the Job dialog box, and when you edit the schedule for any task.

Figure 12-8: Task Schedule Dialog Box: General Tab

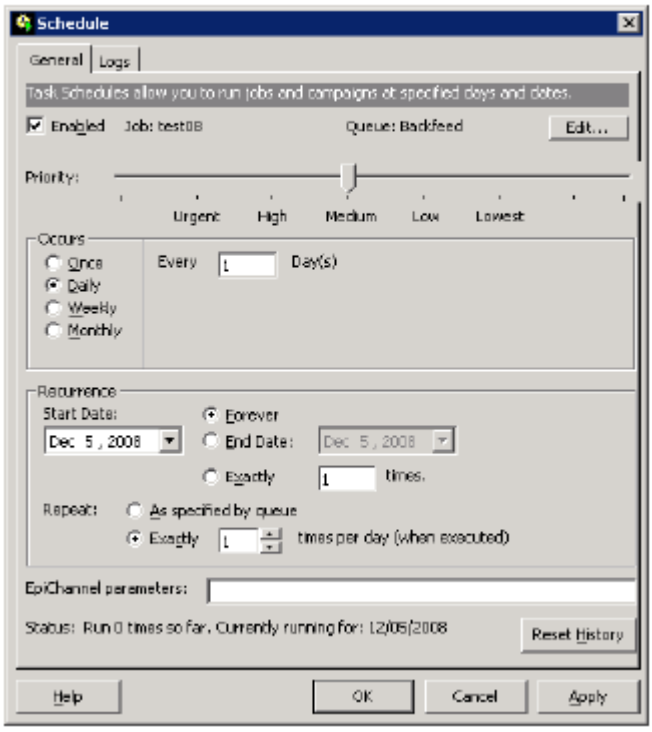


Figure 110: Task Schedule Dialog Box: General Tab

You can use the Task Schedule dialog box to assign a priority and run-time parameters to tasks within a queue. The actual time that the Scheduler runs the queue depends on its relationship to other queues.

Note: Preliminary values for each campaign, chart, or report are assigned based on values specified by the end user. You can change these values in the Task Schedule dialog box.

Scheduling a Task within a Queue

1. Assign this task a priority in relation to other tasks in the same queue. When you move the slider, the integer priority value, which ranges from 0 to 100, is displayed above the slider. A task with a priority of 1 will be executed before a task with a priority of 2.
2. Check **Enabled** to enable the task. Only enabled tasks can be executed.
3. Indicate how often the task occurs (once, daily, weekly, monthly).
4. In the **Recurrence** area set starting and ending dates, or specify the number of times that the task should run. When you specify an exact number of times for it to run, the current number of times actually run is shown in this dialog box after the task has run at least once with this schedule.

5 If you are scheduling an extraction job, use the EpiChannel parameters text box to specify command line parameters. EpiChannel command-line options are described in "The EpiChannel Command Line" on page 351.

6 If you wish to clear the execution history, click **Reset History**. Clearing the execution history resets all of the “times executed” values to zero, but it does not delete the schedule logs for past executions.

If you clear the execution history for a task that was scheduled to start before the current date, the scheduler assumes that all task instances scheduled for past dates are overdue and attempts to execute those task instances if they have not expired.

Viewing Task Logs

To view the task logs, open the **Queue Contents** tab, select the instance of the task, and click **Edit Schedule**. In the Task Schedule dialog box, click **Log** tab (see "Figure 111: Task Schedule Dialog Box: Logs Tab" on page 348) to see the task logs.

Figure 12-9: Task Schedule Dialog Box: Logs Tab

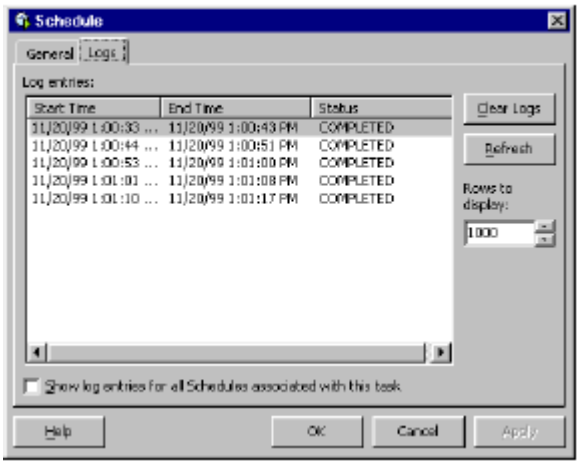


Figure 111: Task Schedule Dialog Box: Logs Tab

The logs shown are for all instances of a task that have been run by this queue or that are about to be run by this queue (that is, all instances that are supposed to run as part of a currently running queue). Some scheduled tasks, such as jobs, can be in multiple queues. When **Show entries for all Schedules associated with this task** is unchecked, only instances associated with this queue are displayed. When the check box is checked, all instances, including ones associated with other queues, are displayed.

Clearing the logs for a queue does not reset the internal execution history. If a daily task has already run, and you clear the logs and re-set the daily run-time for later in the day, the task does not run again at the scheduled time, because it has already run for today.

Predefined Queues

Predefined queues are configured to automatically include the end-user tasks described in Table "Predefined Queues" on page 349. You can change the end-user tasks that are associated with a queue in the **Queue Contents** tab of the Queue dialog box. (See "Figure 108: Queue Dialog Box: Queue Contents Tab" on page 345.)

When a queue wakes up, it finds all saved reports with a role that is associated with the queue and scheduled to run at least once more today, or that have not run as often as they must have on a previous day (but have not yet expired). All of the overdue, but unexpired, task instances are put into the queue, along with one instance of every task that needs to run today.

For example, if a task is supposed to run four times a day, but it ran only once yesterday and not at all today, then (assuming that tasks expire after more than a day), when the queue wakes up, it plans to execute all three of yesterday's missed task executions, plus one more for today.

Running a Campaign queue results in campaign export. Seeds are included in campaign export files by the Scheduler when it runs a campaign from the Campaign queue. See "Campaign Output" on page 198 for information about campaign output files.

Table 62: Predefined Queues

Queue	Description
Charts	When a user creates a favorite chart, that chart must be updated with each extraction to reflect any changed data. These charts become tasks in this built-in queue. Typically, this queue is scheduled to run once after the end of daily extraction and before most users start using the system.
Reports	When a user creates a favorite report, that report must be updated with each extraction to reflect any changed data. These reports become tasks in this built-in queue. Typically, this queue is scheduled to run once after the end of daily extraction and before most users start using the system.
RPDeployments	This built in queue holds tasks that cause the propagation of EpiCenter Profiles, campaigns, and lists to a Real-Time Database for integration with Interaction Advisor. Typically, this queue is scheduled to run once after each extraction update cycle. During development, this queue can be scheduled to run every few minutes so that changes in the attributes in an EpiCenter Profile or list definition can be tested.
Campaign	A task is added to this built in queue whenever a user schedules a campaign export. Typically, this queue is scheduled to run once after the end of daily extraction.

Queue	Description
	Campaign exports can run in parallel because locks are taken on the backfeed tables, to which all campaign exports must write. For campaigns without exclusions, this lock is taken just before information is written to the backfeed tables. For campaigns with exclusions, this lock is taken earlier, when exclusions are computed. This way, the resulting schedule is always serializable. If a backfeed job is running simultaneously, it can also obtain and release the backfeed lock. You can set priority for a scheduled campaign from the Task Schedule dialog box (see also, "Predefined Queues" on page 349). If multiple campaigns have the same priority, there is no precedence among the tasks.
Backfeed	This built-in queue is designed to hold backfeed extraction jobs. Any job that must be scheduled as a backfeed job must be manually added to this queue. Typically, this queue is scheduled to run once a day after the campaign queue has completed so that the Communication information that was added to the backfeed tables is fed back into the EpiMart.
CampaignExtract	This built in queue is designed to hold extraction jobs. No jobs are automatically added to this queue. Typically, this queue is scheduled to run once a day during the system’s least busy time, and after system update cycles complete.
IncrementalExtract	This built in queue is designed to hold incremental extraction jobs. No jobs are automatically added to this queue. Typically, this queue is scheduled to run once a day during the system’s least busy time, and after system update cycles complete.

Running Jobs with EpiChannel

13

Note: Do not copy and paste scripts, commands or code from this document. Line breaks and some other characters picked up when you copy from the PDF can cause errors in the pasted text.

EpiChannel is the Infor Campaign Management extraction program that you run to populate your EpiMart database and to update it on a regular basis. EpiChannel first opens an ODBC or native connection to the source system and then opens a database library connection to the target system and initializes tables. It establishes start and stop date time boundaries and replaces certain special strings. After replacing these strings, EpiChannel executes a user-defined sequence of SQL statements or system calls.

The SQL is pre-processed to match warehouse and database vendor syntax. System calls are pre-processed to supply directory locations and database login information.

Before attempting the first job step, EpiChannel normally verifies the availability of databases and tables (if these options are selected in the **General** tab of Admin Manager's Job dialog box), expands most macros, and runs a limited version of the Scrutiny debugging tool. If these checks fail, the job does not run.

If an error occurs in a job step, EpiChannel can halt, repeat the step, repeat the job, or ignore the error. (See the table ""Possible Actions On Error," on page 9-42," on page 301 for a list of possible actions.) The action taken on error can be configured using Admin Manager.

Whenever a job succeeds or fails, email notification can be automatically sent to the database administrator or any designated list of email recipients. EpiChannel can also be configured to send an email notification when it generates a warning. See "Configuring Email" on page 375 for more information.

The EpiChannel Command Line

This section describes the command-line syntax you can use to invoke EpiChannel at the console's command prompt. These command-line parameters indicate the actions that EpiChannel is to take, identify the databases, and specify execution options. Command-line parameters can specify such options as the maximum selection limit, the interactive debugging mode, and the remote invocation mode. Databases can be specified directly or by reference to an Infor Campaign Management instance.

EpiChannel can be invoked from Admin Manager or from a command line. To invoke EpiChannel from a command line, enter the `epichannel` command at a DOS or UNIX command prompt followed by

the parameters described below. EpiChannel command-line options are not case sensitive, but parameters used in command-line options may be case sensitive, depending on the case sensitivity of your operating system and database engine.

Note: EpiChannel is installed in the bin subdirectory of your Infor Campaign Management installation directory.

Note: Oracle 11g users who are using 64-bit HPUX with the 64-bit Infor Campaign Management application can use the 64-bit Oracle libraries at $ORACLE_HOME/lib. All other Oracle 11g users should use the 32-bit versions of the Oracle libraries when running EpiChannel and other Infor Campaign Management executables. To use the 32-bit libraries, set LD_LIBRARY_PATH (Sun), SHLIB_PATH (HP), or LIBPATH (AIX) to include $ORACLE_HOME/lib32 instead of $ORACLE_HOME/lib.

EpiChannel Command Syntax

The epichannel command runs an extraction job or performs EpiCenter maintenance tasks. Use the following commands, parameters and syntax when working with Epichannel.

```
epichannel [ --help | –h | –? | /? ]

epichannel [ --Job | –j ] job_name job_options general_options

epichannel [ --MomBuild | –mom ] mom_options general_options

epichannel [ --AggBuild | –agg ] general_options

epichannel [ --Scrutiny | –scr ] query_list general_options

epichannel [ --Attach | –a ] instance_number general_options

epichannel [ --CreateCurrentViews | –ccv ] general_options

epichannel [ --Daemon | –dae ] general_options

epichannel [ --Kill | –k ] instance_number general_options

epichannel [ --List | –ls ] list_options general_options

epichannel [ --RandPerm | –rnd ] general_options

epichannel [ --StopDaemon | –stop ] general_options

epichannel [ --GenerateSchema | –gs ] job options
```

The following list describes these commands.

–h, –?, --help, /?	Display detailed on-line help.

–j `job_name` , --Job `job_name`	Execute the job `job_name` , as defined in the EpiMeta database.

The following options can be used for job execution:

–f, --ForceRestart	Force restart. Ignore previous execution history and start executing job from the beginning.
--LogRows num_rows	Log the contents of the first num_rows rows of every extraction command.
–# row_count, -- MaxSelects row_count	Fetch at most row_count rows in any SQL statement.
–t, -- TrialRun	Trial run. Simulate execution of the SQL on the source and destination databases without making any changes to data.
–mom, -- MomBuild	Run MomentumBuilder to build indexes and to generate or update the accelerators required for lists and campaigns.

The following options can be used for MomentumBuilder execution:

–fb, -- ForceRebuild	Rebuild accelerators from scratch. Do not attempt to incrementally update accelerators.
–f, -- ForceRestart	Force restart. Ignore previous execution history and build all accelerators specified in the metadata.
-- NoMirror	Build accelerators in the currently active data mart partition.
–npt num_threads, -- NumParralelThread-snum_threads	Issue up to num_threads concurrent accelerator-creation queries.
–agg, -- AggBuild	Run AggBuilder to build aggregate tables.
–scr `query_list` , -- Scrutiny `query_list`	Run the Scrutiny debugging tool to verify the consistency of the EpiCenter. `query_list` consists of one or more of the following options, separated by spaces
AggBuilder	Run all Scrutiny checks required by AggBuilder.
All	Run all Scrutiny checks.
AppServer	Run all Scrutiny checks required by the Infor Campaign Management Server.
EpiChannel	Run all Scrutiny checks required by EpiChannel.
Mart:Current	Run all data mart Scrutiny checks on the Current tables of the EpiMart database.

Mart:Next	Run all data mart Scrutiny checks on the Next tables of the EpiMart database.
Mart:Prev	Run all data mart Scrutiny checks on the Previous tables of the EpiMart database.
Meta	Run all metadata Scrutiny checks on the EpiMeta database.
MomBuilder	Run all Scrutiny checks required by Momentum-Builder.
Op	Run Scrutiny checks on the EpiOp database.
Schemagen	Run all Scrutiny checks required for adaptive schema generation.
Warning	Run Scrutiny checks for warnings as well as errors.
-- GenerateSchema	Generates your schema according to the settings that you specify. The following options are available for generating your schema from the command line. To use GUI tools instead, or to view more information about the options, see "Generating the Schema" on page 161.
–t, –trialrun	Trial run.
–fb, –forcerebuild	Force rebuild
–rn, –rebuildnecessary	Rebuild when necessary
–at, –adapttable	Adapt table
–bbf, –buildbackfeed	Build backfeed tables
–rnd, –randperm	Build sampling tables
–a `instance_num` , -- Attach `instance_num`	Attach to the execution instance of EpiChannel specified by `instance_num` and print the logs of that instance. You can find the appropriate instance number by running EpiChannel with the --List option.
–ccv, -- CreateCurrentViews	Rebuild the external reporting views that select the currently active versions of base tables.
–dae, -- Daemon	Spawn a daemon process that checks the metadata for remote requests to execute EpiChannel and act on any such requests that it finds.
–k `instance_num` , -- Kill `instance_num`	Kill the execution instance of EpiChannel specified by `instance_num` . You can find the appropriate instance number by running EpiChannel with the --List option.

–ls, -- List	List currently running and pending EpiChannel executions.

The following option can be used when listing EpiChannel executions:

–all, --All	List completed EpiChannel executions in addition to running and pending executions.
–rnd, -- RandPerm	Build the random sampling table that is used to sample demographic tables and to populate random lists.
–stop, -- StopDaemon	Send request to remote daemon to kill all jobs that it has spawned and exit.

The following general options can be used for all applicable EpiChannel command lines:

–b `vendor_name` , -- DBVendor `vendor_name`	The name of the EpiMeta database vendor. Possible values are SQLServer, DB2, and Oracle.
–s `server_name` , -- Server `server_name`	The name of the SQL Server server or Oracle service on which the EpiMeta database resides. On an Oracle database server, this must be a service that is defined in tnsnames.ora.
–u `user_name` , -- UserName `user_name`	The user name for the EpiMeta database. Use the empty argument "" to request the use of operating-system level authentication.
–p `password` , -- Password `password`	The password for the user of the EpiMeta database.
–d `database_name`, -- DBName `database_name`	The name of the EpiMeta database.
–pt `port_number`, -- Port `port_number`	The port number used to connect to the EpiMeta database. Ignored if the connection mechanism does not require a port number.
– sid `oracle_sid`	The Oracle SID of the EpiMeta database. Ignored if the connection mechanism does not require a SID.
-- InstallRootDir `directory`	Specifies the root installation directory of Infor Campaign Management. Note that EpiChannel throws an exception if both --InstallRootDir and --I/--Instance are present. The two options are mutually exclusive.
–i `instance` , -- InstanceName `instance`	The name of a local Infor Campaign Management Server instance. Connection information and other EpiChannel options are read from the local default values for this instance in the registry. Any options that are specified by command-line parameters override these local defaults. Note that EpiChannel

	throws an exception if both --InstallRootDir and --I/--Instance are present. The two options are mutually exclusive.
–r, -- RemoteLaunch	Send request to remote daemon to run the specified EpiChannel command line on the remote server.
–dt, -- Detach	Exit immediately after sending execution request to remote daemon. Do not attach to remote process or print its log file. Can only be used with the --RemoteLaunch option.
–dwc, -- DieWithClient	Request that remote process terminate if connection to client is lost. Can only be used with the --RemoteLaunch option.
–int, -- Interactive	Run EpiChannel in interactive mode. Incompatible with the --RemoteLaunch option.
–nonint, -- Noninteractive	Disable interactive mode for all job steps, including those that are configured to run in interactive mode normally. Incompatible with the --Interactive option.
– DisableDBLog	Disable logging to the EpiOp database. Infor recommends that you do not disable database logging for production extraction jobs. Do not use this parameter with scheduled jobs as it causes the job to fail.
–l `log_dir_path` , -- LoggingDir `log_dir_path`	The directory in which logging files are created.
–sch, -- Scheduler	Specifies that EpiChannel is being called from a scheduler. This option must be used whenever EpiChannel is called from a third-party automated scheduler. Do not use this option when invoking EpiChannel from Admin Manager or a command line, or when scheduling an extraction job using the Infor Campaign Management Scheduler.
–mh `host_name` , -- MailHost `host_name`	The host name of the SMTP host computer that is used to send e-mail alerts.
–ma `email_address` , -- MailAdmin `email_address`	The email address of the administrator for your application. This address is used as the return address for all email notifications.
–ms `email_address` , -- MailSuccess `email_address`	The e-mail address to which EpiChannel log files are sent on successful completion of an extraction job. e-mail_address can consist of multiple addresses separated by commas, without spaces.
–mw `email_address` , -- MailWarning `email_address`	The e-mail address to which a warning is sent when a job completes successfully but generates a warning message. e-mail_address can consist

	of multiple addresses separated by commas, without spaces.
–mf `email_address` , -- MailFailure `email_address`	The e-mail address to which EpiChannel log files are sent on failure of an extraction job. e-mail_address can consist of multiple addresses separated by commas, without spaces.
–mc `character_set` , -- MailMIMECharset `character_set`	EpiChannel uses the character set of your operating system as the default MIME character set for email messages. Use this command if you wish to override the default.

EpiChannel Command Line Examples

The examples in this section illustrate the functionality available through the epichannel command line. They use the following common command line arguments:

-b <database_vendor>	-p <password>
-d <database_name>	-s <server_name>
-j <job_name>	-u <username>
-l <logging_directory_path_name>	

Note: For more examples, and a complete listing of all epichannel command line arguments, access the epichannel command line help function (-h, -?, /?, --help).

Run a Job

The following epichannel example runs job InitialLoad on the metadata in database AS654_epimeta on SQLServer EpnySql (with login sa/secret).

```
epichannel -b sqlserver -s EpnySql -u sa -p secret -d OMtest_epimeta
-l c:\OMtest_logs -j InitialLoad
```

Run a Job Remotely

The following epichannel example runs job InitialLoad remotely, using the instance settings defined in instance OMtest .

```
epichannel -i OMtest -r -j InitialLoad
```

Command line argument -r specifies Remote Launch.

List Jobs

The following epichannel example lists all running jobs on the metadata in database OMtest_epimeta on Oracle server EpnyOra (with login OMtest_epimeta/secret):

```
epichannel -b oracle -s EpnyOra -d OMtest_epimeta -u OMtest_epimeta -
p secret -l c:\OMtest_logs -ls
```

Run Scrutiny on Metadata

The following epichannel examples runs Scrutiny on the metadata in database OM72ME on DB2 server EpnyDb2 (with login OM72/secret):

```
epichannel -b db2 -s EpnyDb2 -d OM72ME -u OM72 -p secret -l c:\OM72_
logs -scr Meta
```

Command line argument -scr Meta specifies that Scrutiny is to be run on the metadata.

Start Epichannel Daemon

The following epichannel examples starts the epichannel daemon, using the instance settings (including meta definition) defined in instance OM720 :

```
epichannel -i OM720 -dae
```

Command line argument -dae spawns an epichannel daemon process that checks for remote requests.

EpiChannel Registry Keys

The EpiChannel program uses several Registry keys, some of the most important of which are shown in "Table 63: Important EpiChannel Registry Keys" on page 359. These keys are located in the following Registry path:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Infor\Instances\
instance_name
\External Interfaces
```

Registry keys for each installed instance of the Infor Campaign Management Server are located in subfolders named after each instance (instance_name).

Table 63: Important EpiChannel Registry Keys

Key	Value
DefaultInstance	Name of default instance
Meta\DatabaseName	Name of EpiMeta database
Meta\DatabaseServer	Name of database server
Meta\DatabaseType	Database vendor and type
Meta\DatabaseUsername	Database login name
Meta\DatabasePassword	Database login password
Email\AdminEmail	Return address for EpiChannel email messages
Email\FailureEmail	Email address for notifications of job failure
Email\MailHost	Mail host for EpiChannel email messages
Email\SuccessEmail	Email address for notifications of successful job execution
Email\WarningEmail	Email address for EpiChannel warning messages

By default, EpiChannel reads the default instance key and uses its value to open the Registry tree that holds all initialization parameters. When a job is run from Admin Manager, EpiChannel is run on the instance that is specified in the Execute Job dialog box. You can also override the default by specifying a different instance name using the **-I** instance_name option.

For example:

```
extract -I
my_instance
```

```
 -J
my_job
```

forces EpiChannel to read all Registry values from the following location:

```
HKEY_CURRENT_USER\SOFTWARE\Infor\Instances\
<my_instance>
```

Output Files

Jobs can log to both files and databases. File logging is always enabled for EpiChannel. Database logging can be disabled by deselecting the **Log to the Database** option in the Execute Job dialog box. The execution of jobs is logged as a unit. Statistics are collected for job steps and for entire jobs. Log files are created in the directory that is specified on the EpiChannel command line or in the Execute Job dialog box.

Extracting New Rows Only

An efficient extraction process pulls only new data and data that has changed since the previous extraction from the source system. The way that the extraction process recognizes this subset of data depends on how the source system identifies changes in data.

Infor Campaign Management supports the following methods of identifying changed data:

- Date or Date/Time fields in source rows
- A source row, or a table it joins to, contains a date or date/time indication of when it is inserted or last updated.
- Timestamp fields in source rows (Microsoft SQL Server only)

  Values can be “stamped” in some way to indicate their status. The EpiCenter provides Microsoft SQL Server timestamp fields for source rows. A source row, or a table it joins to, has a column that contains Microsoft SQL Server data type timestamp . The database engine automatically updates this column whenever the row is inserted or updated. This field does not actually contain any record of the time that the update occurred, but rather values that are incremented whenever a change is specified. If two rows are edited at different times, then the row that is edited more recently has a higher timestamp.
- Highest numeric or string value in named columns in source rows

  A particular column in a particular table is presumed to contain a value that is always increasing from row to row; for example, Order Number or Policy ID. For this method to be useful as a subset

selection criterion, however, either the rows must never be updated, or else the column must be changed by the source system whenever an update occurs.

Using Multiple Readers

You can define multiple parallel readers for each extraction step when extracting source data through EpiChannel.

This is useful in cases where the database and the host machine are performing well, but where EpiChannel is not taking advantage of the available resources - for example, if the CPU usage on the host machine is low during extraction. If this is the case for one or more of your extraction steps, add another reader, and check whether resource usage improves for those steps. Each additional reader requires more memory and processing power, so do not add more than one reader at a time. The optimal number of readers in most cases is between 1 and 4.

Each reader uses its own SQL statement to extract data. Use the following macros to create multiple readers:

- $$NUM_READERS[]
- $$READER_NUM[]

Note: See "“” on page 1" for a discussion of these and other Infor-supplied SQL macros.

These macros can be used as in the following example:

```
SELECT a, b, c FROM table
```

```
WHERE $$READER_NUM[] = $$MODULO[row_number ~,~ $$NUM_READERS[]]
```

The above extraction will ensure that rows are divided uniformly among the readers.

Using `$$READER_NUM[]` in a statement results in the expansion of the statement into `$$NUM_READERS[]` statements that are executed in parallel. Assuming `$$NUM_READERS[]` is three, and that you have a row_number column, a statement such as the one above results in the execution of the following three statements in parallel:

```
SELECT a, b, c FROM table WHE
RE row_number % 3 = 0
```

```
SELECT a, b, c FROM table WHE
RE row_number % 3 = 1
```

```
SELECT a, b, c FROM table WHE
RE row_number % 3 = 2
```

(The example statements are expanded for MS SQL Server, where $$MODULO[] is the % operator. Other databases uses their own modulo operator.)

You define the number of readers in the **Number of Readers** text box in the **SQL Statement** tab of the **Extraction Command** dialog box in Admin Manager. The number you enter there becomes the value returned in `$$NUM_READERS[]`.

Note: It is your responsibility to ensure that the data covered by each reader does not overlap, such as using the $$MODULO[] macro on a unique row identifying column in the source as shown in the prior example.

Configuring Buffer Size

The total amount of memory required for each extraction step is proportional to the number of readers assigned for the step's SQL statement, and to the value set in the BufferSize configuration parameter defined in Admin Manager (under **Configuration Settings > Optimization > EpiChannel > BufferSize**).

The BufferSize configuration parameter sets the number of rows to read at any one time. If you are already nearing the maximum memory usage, but like to experiment with multiple readers, reduce the buffer size setting proportional to the amount of increase for the number of readers.

If your environment has process-level limits on the amount of memory use, keep these limits in mind when increasing the number of readers. For example, a Windows process for EpiChannel with SQL Server would be limited to 2GB of memory use, and a Unix EpiChannel process have limits set by the system administrator.

How EpiChannel Identifies Data To Be Extracted

The following points describe how EpiChannel identifies which subset of data to extract:

- How EpiChannel reads values

  EpiChannel attempts to get a consistent set of dates, timestamps, or column values so that the same single moment in time is captured by the `WHERE` clauses generated by the extraction set

identification macros. EpiChannel minimizes the chance for changes to occur to one set of values, and not to another one, by reading all sets one after another with no intermediate actions. While this approach gives highly consistent values, it is possible that a change can occur between the time EpiChannel reads the first table and column and when it reads the last one, especially if multiple databases are involved.

The values read are stored as the “last” values only if the job commits its work. If any system call or extraction command has an error that aborts the job, it is as if the value stamps have never been read or modified. In the unlikely event that a job aborts during the commit phase, the timestamps can not be synchronized with each other.

The date and timestamp values are read from a table-independent current value such as `SYSDATE` or `GETDATE`.

- Extraction subset identification

  Infor supports extraction subset extraction by reading and recording the current values of extraction-filter fields at the start of the run and remembering these values for the next run. The values are specified available in SQL through the SQL macros described in "Appendix A, “Infor Campaign Management Macros.”" on page 441

- Initializing Extraction Filters

  Before you first run an extraction command that includes extraction filters, you must initialize the extraction filters, as described in "Initializing Extraction Filters" on page 276.

- Disabling Extraction Filters

  You can disable extraction filters for an entire job by checking the **Disable Extraction Filters** check box in the Job dialog box. If this box is checked, all data is extracted for the job without regard to its value range.

  The presence of an extraction set identification macro in an enabled extraction command causes the date, timestamp, or column values referenced in that expression to be read at the start of the extraction job and “memorized” if the job is successful. It does not matter if the macro is in a SQL comment, or even within another Infor Campaign Management macro such as `$$ORACLE[]`. The macro is processed (and the values read) without awareness as to whether the proper parsing of the greater SQL statement indicates the expression is enabled.

  Once a value is memorized, it stays memorized. If you remove the extraction set identification macro for some runs and later re-add it, the values memorized when the expression is last used becomes the “start” range of the re-added macros. If you wish to permanently remove the memorized value, you can delete it in the **Advanced** tab of the Extraction Command dialog box.

  Memorized values are shared between expressions within an extraction step. For example, if two extraction set identification macros reference the value of `current_orders.order_number`, then that value is read and memorized just once. If one of these macros is removed, the other continues to read and update the maximum **order_number**. If the removed macro is then re-added, it is expanded with the updated **order_number** value.

  If all extraction set identification macros that reference `current_orders.order_number` are removed, then the lookup and memorization of **order_number** does not occur. The next macro to reference **current_orders**. **order_number** uses the last value memorized before the macros are removed.

  Note that all dates use the same memorized value, and all timestamps use the same memorized value. Thus, the date value continues to be updated if any extraction-set identification macro

referencing the date remains. Similarly, the timestamp increments as long as any macro referencing timestamp remains.

- Memory for last values

  The “memory” for last values is associated with an extraction step. If the same SQL step is used in two job steps, then those two job steps use the same memory values.

  Note: Since the memorized value is associated with an extraction step, a data store is implicitly associated with that extraction step. Therefore, an SQL extraction step that uses memorized values must not be used with more than one data store.

Aggregate Building

Aggregate building is performed by the AggBuilder step of the EpiChannel extraction program.

The Aggbuilder step builds all newly defined aggregates and updates all aggregates that are invalidated by changes to the underlying fact or dimension table. If a previously built aggregate is not invalidated, then Aggbuilder does not build that aggregate again. If one of the mirrored tables for a fact or dimension is in the Next state, then all aggregates are built or updated on that table. If there is no Next table for a fact or dimension, then aggregates are built or updated in the Current table. See "Swapping Mirrored Tables" on page 192, for a discussion of Current and Next tables. If a dimension aggregate is not used in any fact aggregate, then Aggbuilder does not build that dimension aggregate.

If an extraction job fails while Aggbuilder is running, then aggregates that have already been built successfully are not built again when the job is restarted. The Aggbuilder program builds only those aggregates that need to be built. See "Restartability of Extraction Jobs" on page 181, for a discussion of extraction-job restarting.

The default **Builders** extraction group includes an extraction step named **AggBuild:Agg Builder** that is normally used for aggregate building. You can also invoke AggBuilder using the EpiChannel command line (see "The EpiChannel Command Line" on page 351).

Note: Aggbuilder must be run in any job in which a semantic instance is applied to a fact or dimension table.

Indexes and List and Campaign Optimizations

MomentumBuilder is the extraction step that creates indexes and builds the accelerators (optimized tables and indexes) that are used in generating lists and campaigns. Refer to "Building Indexes and Campaign Accelerators" on page 173, for details. You need to run MomentumBuilder as part of every extraction job.

The recommended order of extraction is as follows:

1 Begin Extraction

2 SQL/Semantics

3 **Builders** . This group includes the following steps:

- AggBuilder
- MomentumBuilder

4 **End of Extraction**

Note: Aggbuilder must be run before MomentumBuilder.

Admin Manager provides a default extraction step called **MomBuild: Momentum Builder** , located in the **Builders** group, that invokes MomentumBuilder. You can also invoke MomentumBuilder using the EpiChannel command line (see "The EpiChannel Command Line" on page 351).

EpiChannel Debugging

The EpiChannel debugging mechanism is thorough and well annotated. EpiChannel provides summary output to the console and creates detailed log files that can be used for debugging.

Job Output

You can display EpiChannel output for debugging purposes on a screen or direct it to a file. The text is displayed on the screen in an 80-column display in a monospaced font, such as Courier. The output is designed to be adequately displayed on low-end monitors.

The job output begins with the exact command line passed to EpiChannel and then shows details of every extraction step, displayed in four columns. The first column shows the type of step, the second column identifies the exact step executed, the third column shows the status, and the fourth column shows detailed statistics (such as execution time and rows extracted). Any warnings or error messages are interspersed with this four-column display.

A portion of a typical job output is shown below:

```
                |                                          |              |
```

```
Group      |ExtractoramaInc/Data          |Started  |@ 7/18/00 7:01:36
PM

           |Extraction/Get Dimension      |         |

           |Data/Individual Extract       |         |

           |                              |         |
```

```
Truncation|ExtractoramaInc/Data          |Started  |@ 7/18/00 7:01:36
PM

          |Extraction/Get Dimension      |         |

          |Data/Individual               |         |

          |Extract/Truncation:           |         |

          |Individual_base               |         |

          |                              |         |
```

```
Truncation|ExtractoramaInc/Data          |Completed|@ 7/18/00 7:01:37
PM

          |Extraction/Get Dimension      |         |

          |Data/Individual               |         |

          |Extract/Truncation:           |         |

          |Individual_base               |         |

          |                              |         |

          |Truncation: Individual_base   |         |Time:

          |                              |         |    500ms

          |                              |         |
```

```
SQL        |ExtractoramaInc/Data          |Started  |@ 7/18/00 7:01:37
PM

           |Extraction/Get Dimension      |         |

           |Data/Individual Extract/SQL: |         |

           |Get Individual (1)            |         |

           |                              |         |
```

```
SQL        |ExtractoramaInc/Data          |Completed|@ 7/18/00 7:01:37
PM

           |Extraction/Get Dimension      |         |

           |Data/Individual Extract/SQL:  |         |

           |Get Individual (1)            |         |

           |                              |         |

           |SQL: Get Individual (1)       |         |Rows:

           |                              |         |    796

           |                              |         |
```

```
          |SQL: Get Individual (1)      |          |Buffer high-water
mark:

          |                             |          |    1

          |                             |          |

```

```
          |SQL: Get Individual (1)       |          |Source-system
execution

          |                              |          |time:

          |                              |          |    70

          |                              |          |

          |SQL: Get Individual (1)       |          |Time:

          |                              |          |    560ms

          |                              |          |

```

All console output is recorded in the epiChannel_summary.log file. (See "Log Directories and Files" on page 374.)

Error Messages

An EpiChannel error message distinguishes between externally generated errors, such as database errors, and internally created messages, such as a file data store that references a nonexistent directory. Each error includes information about the database/file that caused the error, and which specific part of the code detected the error. Most errors contain information about the operations affected or aborted by the error and the metadata objects that specified these operations. Many errors also suggest corrective action.

There are specific error messages for the following:

- `SELECT` statements that do not match columns in the metadata. These display mappings that indicate which return values were missing.
- Rows that fail to insert have their complete set of data displayed.

Note: If your extraction job step calls a stored procedure, use `-exitcmd` in the stored procedure. This ensures that the exit code is passed to the command shell which spawned EpiChannel, and that the error message is accessible to EpiChannel.

Setting Breakpoints

You can use EpiChannel interactively to alter the data fetched by a `SELECT` statement, before the row is forwarded for insertion. You can set breakpoints on extraction commands that change the debug level before the SQL is issued. These breaks can also be associated with particular rows in the return set. For example, it is possible to set a breakpoint that switches to interactive mode just before row 3021 of the **Orders table extraction** SQL statement.

You can use Admin Manager's Extraction Command dialog box to set breakpoints.

Setting Breakpoints to Alter Debugging Levels

1. In the **Advanced** tab, select **Interactive** from the **Debug Level** drop-down list box.
2. Enter the row number at which the debugging goes into effect. You must select the row number immediately before the SQL statement of interest is executed. Enter 0 to begin debugging on the first row. If the row number is not empty the pull/push loop stops fetching before inserting that particular row and waits for permission to proceed. You can alter the data during this pause.

Trial Runs

You can run EpiChannel in trial-run mode, in which SQL is not actually issued against the source or destination systems. This option is recommended for testing the structure of an extraction job.

Running EpiChannel as a Service or Daemon

You can run EpiChannel as a service (on a Windows host) or a daemon (on a UNIX host). This allows EpiChannel to be invoked by a scheduler or by an EpiChannel stub on a remote machine. (See "Running Extraction" on page 174.)

Note: Only one EpiChannel service or daemon can run against a single EpiMeta database at any time.

Note: In order to run a scheduled extraction job, you must have an EpiChannel service or daemon running against the EpiMeta database of the EpiCenter for which you are performing an extraction.

The *Infor Campaign Management Installation Guide* describes the process for installing the EpiChannel service or daemon. If you are running scheduled extractions on the host on which your Infor Campaign Management Server resides, you must use the installer to install the EpiChannel service when you install your Infor Campaign Management Server. To start or stop the EpiChannel service, take the following steps:

1. From the **Start** menu, choose **Settings\Control Panel\Services**.
2. Select the instance EpiChannel service, where instance is the name of your Infor Campaign Management Server instance, from the list of service names.
3. Click **Start** to start this service, or **Stop** to stop it.

For most installations, Infor recommends that you configure the EpiChannel service for automatic startup. To configure the EpiChannel service for automatic startup, take the following steps:

1. From the **Start** menu, choose **Settings\Control Panel\Services**.
2. Select the instance EpiChannel service, where instance is the name of your Infor Campaign Management Server instance, from the list of service names.
3. Click the **Startup** button, choose **Automatic** , and then click **OK** in the Service dialog box to configure automatic start-up for this service.

EpiChannel Output

EpiChannel also generates output in the log directory that you have specified for job execution. Log files are placed in a subdirectory called `epichnl_ timestamp` , where timestamp is a specification of the exact date and time that execution began.

In addition to generating output in the log directory, EpiChannel also logs its actions to special logging tables in the EpiOp data store.

Log Directories and Files

The location specified for a log directory is a parent directory under which a new subdirectory is created for every job. The subdirectories are named based on a timestamp from the start of the EpiChannel run.

EpiChannel generates log files in the log directory that you have specified for job execution. Log files for a job are placed in a subdirectory called jobname , where jobname is the name of the job, such as Generate_Schema. Within this directory, log files for a specific execution of a job are stored in a directory called epichnl_ timestamp , where timestamp is a specification of the exact date and time that execution began. This directory contains two files:

- epiChannel_detailed.log - contains a detailed log of the job execution
- epiChannel_summary.log - contains a summary log of the job execution.

The log directory also contains two subdirectories:

- sql - contains logs of SQL issued during extraction
- threads - contains logs for extraction threads.

The main logging directory also contains a shortcut called Latest that refers to the log directory for the most recent execution of EpiChannel. For each job, the subdirectory for that job contains a shortcut called Latest that refers to the log directory for the most recent execution of that job.

Log Files versus Database Logs

EpiChannel can log to files and databases. Both log files and database logs provide detailed information about job execution. All EpiChannel database logs are created in the EpiOp data store.

You can run EpiChannel with the `--attach` command-line option to attach to a job that is logged to the database in order to view or print the logging information.

Note: A database log is not a log device. A log device is a construct in the database server that logs database transactional activity.

Configuring Email

EpiChannel can automatically send email notification of job success, job failure, and EpiChannel warnings. When running a scheduled job, all such email messages are sent to the address that you entered when installing your Infor Campaign Management instance by default. When running a job from the Execute Job dialog box or from a command line, you can enter distinct email addresses for success, failure, and warning notifications.

If you do not wish to use the same address for all email notifications that are sent from scheduled jobs, or if you wish to change the email configuration, then you must edit the values in the Mail registry key for your Infor Campaign Management instance.

See "The Configuration Dialog Box" on page 147 and to Admin Manager's online help for information on registry key values.

Purging EpiMart and EpiOp Tables

You can remove tables and views that are no longer needed from the EpiMart and EpiOp databases. For example, you can have unused temporary tables or, as a result of metadata changes, you can decrease the number of aggregates. These unused aggregate tables remain in the database until you purge the EpiMart tables.

Note: The Purge utility cannot be run while an extraction job is in progress and must not be run if an extraction job is interrupted and you wish to resume the interrupted job at a later point in time. If you attempt to run the Purge utility when a job is in progress or interrupted, you are given an option to cancel any running job, resetting job history (preventing the resumption of any interrupted job), and

proceeding with the table purge. Unless you are certain that you wish to cancel running jobs and reset job history, Infor recommends that you do not continue with the purge operation in this case.

Figure 13-1: Purge Dialog Box

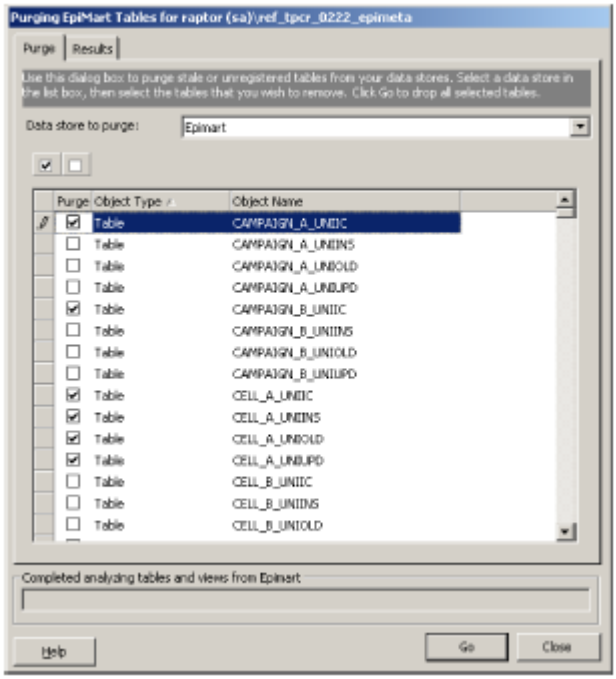


Figure 112: Purge Dialog Box

Purging Database Tables

1 Choose **Purge** from the **EpiCenter** menu to display the Purge dialog box.

Note: All tables that show in the Purge dialog box can be safely deleted from your database. These tables are used during a single run of EpiChannel, as Infor Campaign Management Server temporary tables, or stored information about an aggregate that is deleted. They are not currently needed.

2 Choose a data store from the **Data Store to Purge From** drop-down list box. You can purge tables in the EpiMart data store or the EpiOp data store. When you choose a data store, all purgeable objects in that data store are displayed in the lower pane of the dialog box. These tables are described in "Table 64: Purgeable Tables" on page 377.

3 Check the **Select** check box next to all objects that you wish to purge.

4 Click **Go** to purge the selected tables.

After the run, click the **Results** tab to view the tables that is deleted.

Table 64: Purgeable Tables

Table Name	Description
LIST_*	A materialized form of a saved list. You can save this type of table if you do not want to copy a list from the EpiOp to EpiMart.
TTMMPP_*	Temporary tables that are created by the Infor Campaign Management Server. See "Temporary (TTMMPP) Tables" on page 524 for details on how the Infor Campaign Management Server creates and deletes temporary files.
DimName_##_A	A dimension aggregate table for an aggregate that is deleted, where # is the aggregate number, and A is the data mart letter. Contains a row for every unique tuple in the dimension for the specified columns.
DimName_##_A_p	A partial dimension delta table where ## is the aggregate number of an aggregate that is deleted, and A is the data mart letter.
DimName_##_A_d	A dimension delta table where ## is the aggregate number of an aggregate that is deleted, and A is the data mart letter.
FactName_A##s	A dimension usage table where ## is the dim_role_key, and A is the data mart letter. This table contains a row for each dimension key mentioned in the Suspect table and the new/current values of the dimension columns.
FactName_A##o	An old dimension usage table where ## is the dim_role_key, and A is the data mart letter. This table contains a row for each dimension key mentioned in the Suspect table and contains the old values of the dimension columns.
FactName_A_SUSP	A suspect table where A is the data mart letter. This table contains fact rows that join to at least one dimension row that is updated through a dimension role that is used in at least one aggregate, and all inserted fact rows (from yesterday and today).
KMU_###_##	A dimension usage keymap table where ### is the dimension role, and ## is the dimension aggregate number for an aggregate that is deleted. This table maps every row in the dimension usage table to a row in the dimension aggregate.

Table Name	Description
KMU_###_##_old	An old dimension usage keymap table where ### is the dimension role, and ## is the dimension aggregate number for an aggregate that is deleted. This table maps every row in the old usage table to a row in the dimension aggregate.
KM_###_##_#	A dimension keymap table where ### is the dim_base_key, ## is the source aggregate number, and # is the destination aggregate number. At least one of the aggregates is deleted. This table maps every row in the source dimension aggregate to a row in the destination dimension aggregate.
FactName_##_A	A table for an aggregate that is deleted, where ## is the aggregate number, and A is the data mart letter.
FactName_##_A_d	A fact delta table where ## is the aggregate number for an aggregate that is deleted, and A is the data mart letter This table is an aggregate of the Suspect table.

Toggling Data Mart Tables

You can use the Job Control dialog box to roll back, cancel and commit jobs. To display the Job Control dialog box, click **Job Control** on Admin Manager's tool bar.

Figure 13-2: The Job Control Dialog Box

Figure 113: The Job Control Dialog Box

To switch to the set of mirrored tables that was active before the most recent extraction, click **Roll Back**. When you click this button, the state of the A/B tables for all facts and dimensions is toggled back to the previous state of these tables. The state of the backfeed tables is not changed. To switch

back to the set of mirrored tables that was made active by the most recent extraction, click **Commit**. To cancel a job, roll back the job and then click **Cancel**. You cannot commit a job after you have cancelled it.

Queries run against the set of tables that are enabled (the Current tables), and extractions populate the other set of tables. See "Data Mart Mirroring: A and B Tables" on page 191 for more information on mirrored data mart tables.

Backing up the EpiOp Database

The EpiOp database for your EpiCenter contains important user-generated data, such as saved lists and information about campaigns that are executed. Infor recommends that you use your database administration tools to back up the EpiOp database on a regular basis.

Issuing SQL Queries

You can issue SQL statements against the EpiMeta, EpiMart, or EpiOp database from the SQL Query Window. To open the SQL Query Window (see "Figure 114: The SQL Query Window" on page 379), select **SQL Query Window** from the **Tools** submenu of the **EpiCenter** menu.

Figure 13-3: The SQL Query Window

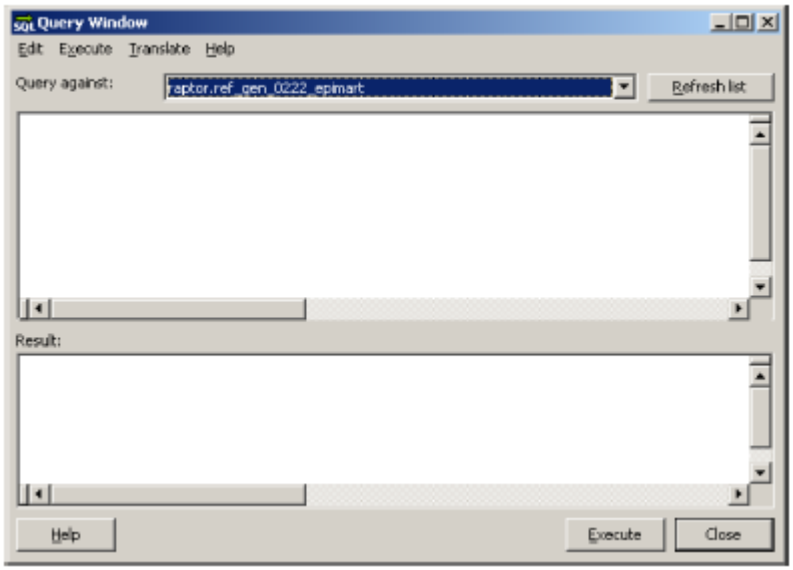


Figure 114: The SQL Query Window

Select the database against which you wish to issue queries from the **Query** radio buttons. To execute a query, enter the query in the upper pane and select **Execute All** or **Execute Selection** from the **Execute** menu. Query results are displayed in the lower pane.

You can also use the SQL Query Window to translate statements that contain Infor Campaign Management database-independent macros (see "Appendix A, “Infor Campaign Management Macros”" on page 441) into database-specific statements. Enter the statement that you wish to translate in the upper pane and choose **Translate All** or **Translate Selection** from the **Translate** menu. The translation for the database that you have selected from the **Query** radio buttons is displayed in the lower pane.

Running Queries Against Data Mart Tables

The **End of Extraction** extraction group includes the **Create Current Views** extraction step. This step creates views that contain current information in the data mart as of the last extraction. You can use these views to provide access to data mart data for queries that originate outside of your Infor Campaign Management applications. See "Running Queries Against Data Mart Tables" on page 384 for more information.

Infor Campaign Management Utilities

14

Note: Do not copy and paste scripts, commands or code from this document. Line breaks and some other characters picked up when you copy from the PDF can cause errors in the pasted text.

This chapter describes utilities that provide remote access and command-line access to some of the data mart- and application-configuration functions that Infor Campaign Management provides. These utilities include:

- Restricted versions of Admin Manager:
  - User Interface Only
  - Security Only
  - Output Processors Only
- The SQL Query Window
- The Scrutiny validation utility
- The EpiCenter report utility
- The Alter Object Properties utility
- The Instance Management utility
- Log Truncation tools
- Admin Manager command line
- Admin Manager scripting interface
- Mart Management commands

The sections that follow describe these utilities.

Restricted Versions of Admin Manager

To allow for administrative roles, and to provide restricted access for selected users of Admin Manager, Infor supports restricted versions of Admin Manager. These restricted versions display only a subset of the folders that appear in the complete version. In all other respects, the restricted versions operate in the same manner as the full version.

Access to the full version of Admin Manager, and to restricted versions, is governed by a set of version-control files that reside in the ConfigFiles\ver subdirectory of the installation directory for your Infor Campaign Management instance. You control access to Admin Manager versions by granting or removing read permission on the following files:

- Admin.ver

  Read-access permission on this file grants access to the full-function version of Admin Manager.

- Output.ver

  Read-access permission on this file grants access to the Output Processors Only version, which displays the contents of the Output Processors folder.

- Security.ver

  Read-access permission on this file grants access to the **Security Only** version, which displays the contents of the Users and Groups folders.

- UI.ver

  Read-access permission on this file grants access to the **User Interface Only** version, which displays the contents of the Measures and Presentation folders.

If you grant read permission on Admin.ver , users have access to the full functionality of Admin Manager. If you revoke read access to that file, and then grant read access to any of the other .ver files, users have access only to the folders that are controlled by those files. For instance, if you grant read permission to a group called EpiApps for the Output.ver , Security.ver , and UI.ver files, users in that group can perform the tasks that are involved in maintaining an Infor Campaign Management application. Those users can not able to perform changes to the data mart or the extraction process.

You can also use the following command-line options to invoke restricted versions of Admin Manager:

- `EpiMgr.exe -o`

  Grants access to the Output Processors folder

- `EpiMgr.exe -s`

  Grants access to the Users and Groups folders

- `EpiMgr.exe -u`

  Grants access to the Measures and Presentation folders

If you include any of the above options on the command line, Admin Manager only gives you access to the folders that correspond to the options you specify. Command-line options do not override the restrictions imposed by .ver files. For example, if you invoke `EpiMgr -os` but you do not have read permission on Security.ver , only the Output Processors folder appears.

The SQL Query Window

You can issue SQL statements against the data stores in your EpiCenter and in other EpiCenters from the SQL Query Window. To open the SQL Query Window (see "Figure 115: The SQL Query Window" on page 383), select **Tools > SQL Query Window** .

Figure 14-1: The SQL Query Window

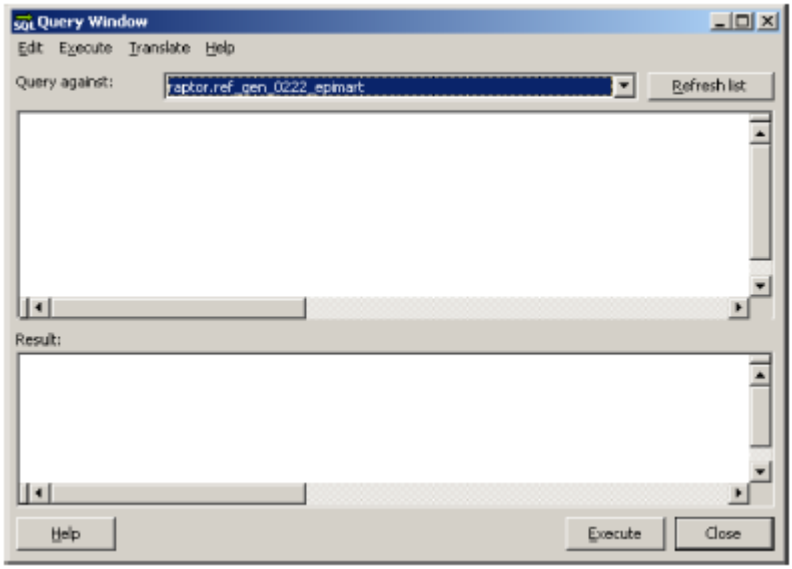


Figure 115: The SQL Query Window

Select the database against which you wish to issue queries from the **Query Against** list box. This list box contains the following items:

- all built-in data stores (EpiMeta, EpiMart, and EpiOp)
- all other data stores except ODBC and File stores
- all other databases on the same server(s) as your EpiMeta, EpiMart, and EpiOp databases

The last option in the **Query Against** list box is **(Refresh List)** . This option requires the servers to get an updated list of databases.

To execute a query, enter the query in the upper pane and select **Execute All** or **Execute Selection** from the **Execute** menu. Query results are displayed in the lower pane.

You can also use the SQL Query Window to translate statements that contain Admin Manager database-independent macros (see "Appendix A, “Admin Manager Macros”" on page 441) into database-specific statements. Enter the statement that you wish to translate in the upper pane and choose **Translate All** or **Translate Selection** from the **Translate** menu. The translation for the database that you have selected from the **Query** radio buttons is displayed in the lower pane.

Running Queries Against Data Mart Tables

The **End of Extraction** extraction group includes the **Create Current Views** extraction step. This step creates views that contain current information in the data mart as of the last extraction. You can use these views to provide access to data mart data for queries that originate outside of your Infor Campaign Management applications. Available views are listed in "Views" on page 527.

Note: Current views are not up-to-date if a database query is created while the Create Current Views extraction group is running during the last extraction job.

As discussed in "Views for External Reporting" on page 197, the names of these views in the EpiMart database have the suffixes `_0_V`, `_0_RV`, and `_CV`. You must direct all external queries for data mart data against these views. To perform aggregate operations on fact columns, use the `SUM` operator. When you want to break out subtotals by attribute values, use `GROUP BY` operations on dimension columns. When you want to perform a join between fact and dimension, data, use the appropriate dimension role columns, as shown in the following example:

```
SELECT

SUM(fact.my_fact_col),

dim.
my_dim_col


FROM

my_fact
_0_V fact,
my_dim
_0_V dim

WHERE

fact.
my_dim_role
_key = dim.
my_dim
_key

GROUP BY

dim.
my_dim_col
```

Here, my_fact is the fact table being queried and my_fact_col is a column of that fact table. my_dim_role is a dimension role that is included in the dimensionality of my_fact , my_dim is the base dimension to which my_dim_role refers, and my_dim_col is a column of that dimension.

If you wish to use the _CV integer map view for an integer mapped column, you can find the name for the desired view by using the SQL Query Window (see "The SQL Query Window" on page 383) to translate the following macro code:

```
$$DIM_COL_INTMAP_VIEW[
dim_name
~,~
col_name
]
```

Here col_name is the name of the integer-mapped column and dim_name is the name of the dimension in which the column appears.

You can also issue queries against built-in views in the EpiMeta database to find available fact columns, dimension columns, dimension roles, and degenerate dimensions.

The fact_col_view view shows information about all fact-table columns. The following example displays a lists of fact columns:

```
SELECT

fact_tbl_name, fact_col_name

FROM

fact_col_view

ORDER BY

fact_tbl_name
```

The dim_col_view view shows information about all dimension-table columns. The following example displays a lists of dimension columns:

```
SELECT

dim_base_name, dim_col_name

FROM

dim_col_view

ORDER BY
```

```
dim_base_name
```

The `fact_dim_role_view` view shows information about all dimension roles that are associated with each fact table. The following example displays a list of dimension-role keys for each fact table, along with the associated dimension key:

```
SELECT
fact_tbl_name, dim_epikey_role, dim_epikey
FROM
fact_dim_role_view
ORDER BY
fact_tbl_name
```

The `fact_degen_view` view shows information about all degenerate dimensions that are associated with each fact table. The following example displays a lists of degenerate dimensions for each fact table:

```
SELECT
fact_tbl_name, degen_key
FROM
fact_degen_view
ORDER BY
fact_tbl_name
```

The Scrutiny Debugging Tool

Scrutiny is an interactive debugging tool available from Admin Manager's **EpiCenter** menu. Scrutiny provides output that you can examine to find potential problems in your EpiCenter. Scrutiny issues

both errors, indicating that a component is likely to fail if used, or warnings, indicating that the component will work, but that something is configured in a suboptimal way.

The Infor Campaign Management Server runs a subset of Scrutiny checks whenever you start your Infor Campaign Management application, and the Infor Campaign Management Server does not start if any of these checks fail. By default, EpiChannel runs a subset of Scrutiny checks when performing an extraction , and extraction fails if any of these Scrutiny checks fail.

Some Scrutiny checks are also run during schema generation. If any of these automatic Scrutiny checks fail, you can use Scrutiny from within Admin Manager or from the command line to diagnose and correct the problem.

Note: Infor recommends that you run Scrutiny with only the checks needed for schema generation before running Generate Schema. This early run with Scrutiny will allow you to find and correct problems more conveniently.

Scrutiny ensures that your EpiCenter is in a consistent and functioning state, by running an extensive set of queries against your EpiCenter. Scrutiny can check both EpiMeta and EpiMart databases to ensure that various rules are followed and that object descriptions are mutually consistent.

For example, Scrutiny can examine the EpiMart tables for such problems as missing indexes, missing rows, and bad referential integrity. It can also find violations of list-management constraints in the EpiMeta database, and checks consistency constraints, such as the presence of `UNKNOWN` rows in EpiMart tables and the use of well-formed references.

	Statistics are collected for EpiMeta and EpiOp databases to improve the performance of Scrutiny queries and function-based indices. Statistics are collected after EpiMeta and EpiOp initialization, Generate Schema, upgrade, and interactive metadata import. You should regularly refresh these statistics to keep them up to date. Two new tools are provided for this task in the Tools menu: • **Tools > Miscellaneous > Refresh Meta Statistics (EpiMeta)** • **Tools > EpiOp > Refresh Op Meta Statistics (EpiOp)** You can also configure your own refresh scripts with the following scripting commands: • import_metadata REFRESH_META_STATS • Refreshes database statistics for EpiMeta objects at the end of import. • refresh_epimeta_stats • Refreshes database statistics for EpiMeta objects. • refresh_epiop_meta_stats Refreshes database statistics for EpiOp objects.

Running Scrutiny from within Admin Manager

1 Choose **Run Scrutiny** from the **EpiCenter** menu.

2 In the Scrutiny dialog box, choose the databases (**EpiMart**, **EpiMeta**, or both) to check.

Figure 14-2: Scrutiny Debugging Tool

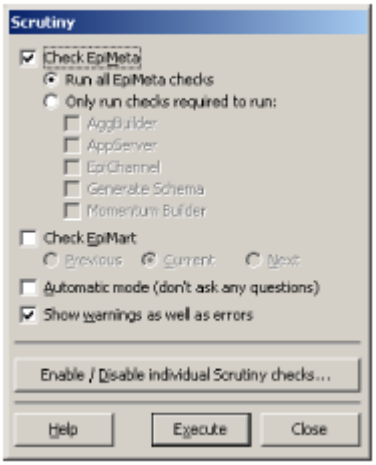


Figure 116: Scrutiny Debugging Tool

3 If you are checking the EpiMeta tables, you can choose to perform all consistency checks or only those checks that are required by selected Infor Campaign Management utilities.

4 If you are checking the EpiMart database, select the set of mirrored tables to check. The current tables are selected by default. (See "Data Mart Mirroring: A and B Tables" on page 191.)

5 Check **Automatic Mode** to perform all checks non-interactively without fixing any problems that are encountered. This option is useful if you wish to generate a list of all errors that are currently present.

6 To display warnings about conditions that are unusual but does not cause your Infor Campaign Management applications to fail, check **Show Warnings as Well as Errors** . Infor recommends that you leave this option selected.

7 To enable or disable specific Scrutiny checks, click **Enable/Disable Specific Scrutiny Checks** and then select or deselect specific checks in the Choose Scrutiny Checks to Enable dialog box (see "Figure 117: The Choose Scrutiny Checks to Enable Dialog Box" on page 389).

Figure 14-3: The Choose Scrutiny Checks to Enable Dialog Box

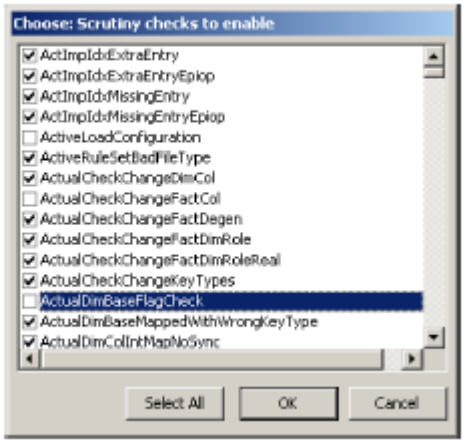


Figure 117: The Choose Scrutiny Checks to Enable Dialog Box

Note: If you disable a Scrutiny check, the check remains disabled until you reenable it.

8 Click **Execute** to run Scrutiny.

In general, Scrutiny runs quickly (a few minutes for large EpiCenters), especially when **Check EpiMart** is not selected.

Scrutiny displays a text screen that shows the checks it is running. If an error is detected, Scrutiny describes the nature of the error and offers suggestions about how to fix it yourself (usually in Admin Manager) or offers to correct the error for you. If you like Scrutiny to execute the proposed solution, press **y** . After Scrutiny has executed, it asks if you want to rerun that section to ensure that the fix was applied successfully.

Admin Manager

Note: In some cases, Scrutiny fixes are last resorts and must be applied only if all else fails. These checks are identified as such, but be sure to read the descriptions carefully before applying any fix.

Scrutiny logfiles are placed in the logging directory that is specified in the **General** tab of the Preferences dialog box, available from Admin Manager's **File** menu.

Admin Manager

The EpiCenter Report Utility

You use the EpiCenter Report utility to create a report of all or part of your EpiCenter metadata configuration. This report is an HTML file that is viewable in your browser window (as shown in "Figure 118: An EpiCenter Report" on page 390).

You can generate this report whenever you need a snapshot of your current EpiCenter. You can use your web browser to open saved reports for viewing. For example, you need to generate reports while you are developing your EpiCenter in order to record your progress.

Figure 14-4: An EpiCenter Report

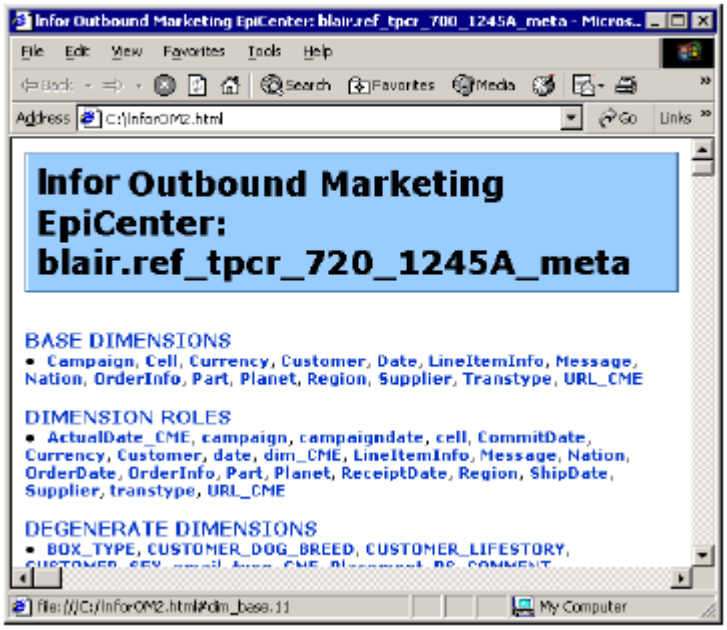


Figure 118: An EpiCenter Report

Generating an EpiCenter Report

1 Do one of the following to open the Report dialog box:

- Select an EpiCenter in Admin Manager and choose **Report** from the **EpiCenter** menu.
- Right-click the folder for an EpiCenter and select **Report** from the pop-up menu.

Figure 14-5: EpiCenter Report Dialog Box

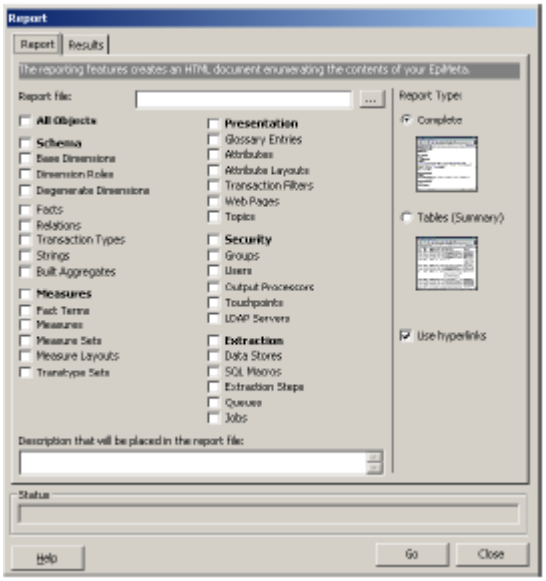


Figure 119: EpiCenter Report Dialog Box

2 Enter the **Report File** name preceded by the directory location. If you do not enter this information now, you are requested to do so later. If the **Report File** name is the same as the name of an existing file, then that file is overwritten.

3 Select all or a subset of the schema objects.

4 Select a **Report Type**. A **Complete** report gives a complete report of all metadata. A **Tables** report shows tables that provide a summary of metadata. "Figure 118: An EpiCenter Report" on page 390 shows a section of a complete report.

5 Click **Go** to generate the report. By default, the report appears in your web browser after the file is generated and hyperlinks are created. You can disable hyperlinks to save time.

6 Click the **Results** tab of the Report dialog box to view the status and elapsed time of the report-generation process.

The Alter Object Properties Utility

If you change the physical object properties of a schema object (see "Physical Object Properties" on page 92, "Base Dimension Tables" on page 100 and page "Configuring Object Properties" on page 137, and "Appendix E, “Object Types and Views”" on page 517), those changes are not automatically propagated to existing objects in your EpiMart database. Changes are propagated to any objects that are completely rebuilt.

Some changes can be propagated using the **Alter Object Properties** utility. This utility can change the following properties for existing tables and indexes:

- Degree of parallelism for queries

- Extent size (changes applied to new extents only)

Note: Extent size cannot be changed in locally managed tablespaces.

In addition, the utility can change the following properties for tables:

- Percent free (changes applied to new blocks only)
- Percent used (changes applied to new blocks only)

Note: Partition-level overrides for Extent Size, Pctfree, and Pctused (where applicable) are not applied in the Alter Object Properties utility.

To use the **Alter Object Properties** utility, select **Alter Object Properties** from the **Tools** menu for your EpiCenter. In the Alter Object Properties dialog box, check **Trial Run** to do a trial run that makes your changes and click **Go** . If the trial run completes successfully, deselect **Trial Run** and click **Go** to propagate your changes in object properties to the EpiMart database.

The Instance Management Utility

You use the Infor Instance Manager to create, edit, and delete instances of Infor Campaign Management on your local machine. The Installer installs files that are shared by all instances, while the Instance Manager wizard enables you to manipulate the files that are instance-specific.

See “Configuring the Infor Campaign Management Instance“ in chapter 8 in the *Infor Campaign Management Installation Guide*, for instructions on how to access and use the Infor Campaign Management Instance Manager.

Log Truncation Tools

You can use Admin Manager to truncate the extraction logs and the query logs in your EpiOp database.

To permanently remove all data in the extraction logs, select **EpiOp > Truncate Extraction Logs** from the **Tools** menu for your EpiCenter.

Note: Truncating extraction logs removes all extraction information that are logged to the EpiOp database. Extraction log files are not affected by this operation.

To permanently remove all data in the query logs, select **EpiOp > Truncate Query Logs** from the **Tools** menu for your EpiCenter.

The Infor Campaign Management Server writes additional logs in the `<` instance `>\Infor Campaign Management\logs` directory. For example, if your Infor Campaign Management Server instance directory is the following:

- `<Infor Campaign Management Install directory>\Web\OM1000`

then, your logs are in the following directory:

- `<Infor Campaign Management Install directory>\Web\OM1000\AP\logs`

You can control how long files stay in this directory through the **Optimization/Query/TempDirGarbageLifetime** configuration setting. The value of this setting is interpreted as a number of seconds. The default value is 604800 seconds, or 1 week. For more information on Infor Campaign Management Server logs, see “Logging” in chapter 10 of the *Topic Implementation Guide*.

Admin Manager’s Command Line

You can invoke the main Admin Manager user interface or Admin Manager’s scripting interface from the Windows command line. (See "Admin Manager Scripting Interface" on page 396.) To do so, open a command prompt and enter a command of the form: `<instance_directory>\bin\epimgr.exe arguments`

Where instance_directory is the directory path to your Infor Campaign Management instance and arguments are a subset of the arguments described in "Table 65: Admin Manager Command-Line Arguments" on page 394.

For example, if you installed Infor Campaign Management into the default installation directory and you enter the following command: `C:\Program Files\Infor\ Campaign Management\10.1.0\bin\epimgr.exe -o-locale fr`

The Output Only form of the windows-based Admin Manager interface opens in French (if the French language is installed).

For the most up-to-date information about this utility, refer to:

<installation_directory>\ConfigFiles\Docs\<locale>\APMgr_args.txt

where <installation_directory> is the path to your Infor installation directory, and <locale> is the two-letter code for the language in which you wish to view the file.

The epimgr command invokes either the main windows interface or the scripting interface of Admin Manager. It takes the following form:

```
epimgr.exe [ -u | -s | -o | -version | -locale <
locale code
> | -configdir <
dirname
> | -logdir <
dirname
> | -epichannel <
dirname
> | -showargs | -batch | -f <
scriptname
> [<
```

```
logfilename
>] | -fc <
script command
> | -fl <
logfilename
> ]
```

The following list describes the command line arguments that you can use with the **epimgr** command.

Table 65: Admin Manager Command-Line Arguments

- u	Invokes Admin Manager in UI Only mode. See Also, "Restricted Versions of Admin Manager" on page 381 for information.
-s	Invokes Admin Manager in Security Only mode. See "Restricted Versions of Admin Manager" on page 381 for information.
-o	Invokes Admin Manager in Output Only mode. See "Restricted Versions of Admin Manager" on page 381 for information.
- version	Displays Admin Manager's version number and quits.
- locale <locale code>	Sets the user interface and help language for the current session only. The default setting is taken from the registry: `Software/Infor/EpiCenterManager/<version>/Preferences/UILocale` You can change the default locale in Admin Manager by selecting File > Preferences.
- configdir <dirname>	Sets the configuration directory value for the current session of Admin Manager only. This directory is used as a reference for relative paths to EpiChannel, scripts, and any other executables that are included in your installation. The default configuration directory is taken from the registry: `Software/Infor/EpiCenterManager/<version>/Preferences/ConfigFileDir`

	This value changes whenever a new instance of Admin Manager is installed, even if the new instance is from the same version of Infor Campaign Management as the original.
- logdir <dirname>	Sets the extraction job logging directory for the current session of Admin Manager only. The default logging directory is <installationDir>\JobLog, where installationDir is the path to your Infor Campaign Management instance.
- epichannel <dirname>	Sets the path to the epichannel executable to dirname.
- showargs	Displays the arguments as they are parsed by Admin Manager. This argument allows you to diagnose argument errors that arise from being passed through DOS and other scripts.
- batch	Invokes the batch mode of Admin Manager. In batch mode, interactive messages do not appear while script commands are being processed. Infor provides an additional command called EpiMgrLaunch. This command passes a command line to epimgr.exe that has -batch as the first argument followed by any arguments that are used in the EpiMgrLaunch command line.
- f <scriptname>	Invokes Admin Manager in batch mode and runs the script specified by scriptname. The resulting logfile is named scriptname if not otherwise specified by the -fl argument. You can use only one -f argument in a single command line. An -f argument cannot be combined with a -fc argument. See "Admin Manager Scripting Interface" on page 396 for additional information.
-fc <command>	Invokes Admin Manager in batch mode and executes the script commands specified by command. Unless the -fl argument is supplied, the resulting logfiles are saved in <LoggingDir>\<generatedSubDirName>\Command_Line.log. You can use multiple -fc arguments in a single command line. An -fc argument cannot be combined with a -f argument. See "Admin Manager Scripting Interface" on page 396 for additional information.

-fl <logfilename>	Sets the name of the logging file: • If logfilename is supplied with a full directory path, Admin Manager logs to that directory path, even if it is not in the normal logging directory. • If logfilename is supplied without a directory path, Admin Manager logs to <LoggingDir>\generatedSubDirName\logfilename.

Admin Manager Scripting Interface

Admin Manager also includes a scripting interface that allows you to automate EpiCenter maintenance activities. Scripts can be called from some Admin Manager dialog boxes and menus, and from the Windows command line. Infor provides several built-in scripts for common maintenance activities, and you can also configure additional scripts for maintenance activities that apply to your installation.

The following list describes the types of scripts that you can use with Admin Manager:

EpiScripts	Scripts expressed in the EpiScript syntax described in the following sections. These scripts end in *.epi.
SQL scripts	Scripts expressed in plain SQL. These scripts end in *.sql.
Enhanced SQL scripts	Scripts expressed in SQL that uses Infor macros and where phrases are pre-processed and replaced. These scripts end in *.sql.

Note: EpiScripts may include the execution of SQL scripts, but, due to the limitations of the database vendors, SQL scripts cannot invoke EpiScripts.

Executing Scripts within Admin Manager

Built-in scripts are available from Admin Manager's dialog boxes and menus listed in the table below. The names of the script choices that are displayed in these menus and dialog boxes are taken from the names of the *.sql files contained in the specified directory. Each script is documented with comments inside the script itself.

To add additional scripts to these dialog boxes and menus, copy them into the script directory specified in the table. You can use the built-in scripts as templates for scripts that you configure for your own installation.

Table 66: Script Locations in Admin Manager's Windows Interface

Admin Manager Commands that are located in this area...	...are based on scripts in this directory.
Initialize EpiCenter dialog box (available from the EpiCenter menu)	<InstallDir>\ConfigFiles\Configurations\init
Topic dialog box (available in the Presentation folder)	<InstallDir>\ConfigFiles\UpdateScripts\Topic
Tools menu > EpiMart	<InstallDir>\ConfigFiles\UpdateScripts\Main-Screen\EpiMart
Tools menu > EpiOp	<InstallDir>\ConfigFiles\UpdateScripts\Main-Screen\EpiOp
Tools menu > Internationalization	<InstallDir>\ConfigFiles\UpdateScripts\Main-Screen\I18n
Tools menu > Miscellaneous	<InstallDir>\ConfigFiles\UpdateScripts\Main-Screen\Miscellaneous
Tools menu > Topic	<InstallDir>\ConfigFiles\UpdateScripts\Main-Screen\Topic
Tools menu > Upgrade	<InstallDir>\ConfigFiles\UpdateScripts\Main-Screen\Upgrade

Executing Scripts from the Command Line

You can also execute scripts and individual script commands from the command line. There are two forms:

-f <scriptname>	Enter batch mode and execute the named EpiScript.
-fc <script command>	Enter batch mode and execute the script command along with any other -fc script commands.

Executing Episcripts

To execute an EpiScript from the command line, use: `EpiMgrLaunch.exe -f <scriptname>`

The script must be an EpiScript. SQL scripts are used only by having an EpiScript command execute them.

Note: Do not use EpiMgr.exe to execute a script. EpiMgr.exe executes in the background and does not return messages or exit codes. Your command line returns immediately without waiting for EpiMgr.

EpiMgrLaunch.exe passes all of its command line arguments to the EpiMgr.exe found in the same directory, waits for EpiMgr to finish, and returns the exit code and any messages to the operating system.

Executing Individual Script Commands

To execute one or more individual script commands, use: `EpiMgrLaunch.exe -fc <scriptcommand>`

Note: You can use multiple -fc arguments in a single command line, but -f arguments must be unique and cannot be combined with other -fc commands in the same command line. Multiple -fc arguments are evaluated from left to right

The scriptname parameter can include a directory path. If a directory path is not specified, Admin Manager searches the ConfigFiles directory of the most current version of Admin Manager. If you wish to change this directory, you can use the `-configdir` argument in Admin Manager's command line.

Each `-fc` command acts as if it is a line given in an EpiScript file made up of all of the -fc commands. For example:

EpiMgrLaunch.exe -fc "connect_meta SQLServer my_server my_epimeta my_username my_password" -fc "export_metadata_add_sequence Macro \"translation_string in ('ALTER_TABLE', 'NUMBER')\" " -fc "export_metadata NewMacros.mdb"

is equivalent to a file with the following commands:

- `connect_meta SQLServer my_server my_epimeta my_username my_password`
- `export_metadata_add_sequence Semantic "semantic_type_name in ('First Dimension Value')" "`
- `export_metadata_add_sequence template \ "adaptive_template_name in ('index_dim ')"`
- `export_metadata NewSemantics.mdb`

To execute such a file, use `EpiMgrLaunch.exe -f <filename>` . Using -fc as described above eliminates the use of temporary script files, but makes for more complex command lines.

Script Errors

Any error in an EpiScript halts further processing of the EpiScript. However, some EpiScript commands have the ability to not consider an error in their execution to be an EpiScript error. The database vendor determines whether errors in SQL scripts halt further execution of the SQL script. In practice, most SQL errors halt only their execution block. Non-ignored errors in SQL scripts fired by EpiScripts become errors in the EpiScript and halt further EpiScript processing.

Script Logging

Both the -f and the -fc commands automatically create a subdirectory in the current logging directory where all script operations are logged:

- A directory created with the `-f` argument is named for the script that is executed.
- The directory created with the `-fc` argument is named Command_line.log .

You can specify your own logfile name by using the `-fl` argument.

Scripting Syntax

Note: Consult APMgr_args.txt for additional documentation on scripting syntax. It is stored in the following directory:

```
<install_directory>/ConfigFiles/Docs/<locale>/APMgr_args.txt
```

Scripts can be written in any plain-text editor. Include each command on a single line with command arguments separated by one or more spaces. To skip an argument, use a pair of double quotes. For example:

command arg1 “arg2“ arg3 ““ arg5

Command arguments that contain a space (such as a file name) can be quoted with double-quotes. For example: `include "C:\Program Files\Infor\Campaign Management\10.1.0\ConfigFiles\Configurations\utilities\BasicInit.epi"`

Command arguments that contain a quote can escape the quote character with a backslash (\). For example: `include "C:\Program Files\Infor\Campaign Management\10.1.0\ConfigFiles\file_name_with_a_\"quote\"_in_it"`

Commands and special words in arguments are case insensitive.

By convention, most script filenames end in “.epi,” though this suffix is not required for proper operation.

Note: Tab characters are not recognized as whitespace. Do not use tabs instead of spaces in scripts.

Scripting Macros

EpiScripts and expanded SQL scripts can contain macros (expansion variables proceeded by $$) that Admin Manager replaces with actual content before interpreting the command. "Table 67: Scripting Macros" on page 401 describes the Infor scripting macros.

Using the Set Command

Use the set command to set the value of the expansion variable. There are three forms of this command, each is discussed in the following sections:

- `set <EPISCRIPT_EXPANSION_VARIABLE> <NEW_VALUE>`

- `set <EPISCRIPT_EXPANSION_VARIABLE> From_Configuration_Setting <CONFIG_PATH>`
- `set USE_FUNC_IDX [0|1]`

Note: Script execution causes the initial value of an expansion variable to change.

Setting a Macro to a Constant Value

The first form of the `set` command works in any EpiScript and sets an existing or a new expansion variable to a constant value. It takes the following form:

```
set <EPISCRIPT_EXPANSION_VARIABLE> <NEW_VALUE>
```

For example:

```
set TRANSSET Custom_AS
set Translation_Root e:/Apps/MyStuff
```

With these, the line

```
print TRANSET = $$TRANSSET and Translation_Root is $$Translation_Root
```

is treated as

```
Print TRANSET = Custom_AS And Translation_Root is e:/Apps/MyStuff
```

Setting a Macro to an Existing Value

The second form of the `set` command is valid only when connected to an initialized EpiMeta and sets an existing or new expansion variable to the value currently held in Configuration/Settings for that path. (You must use quotes if the path contains spaces.) It takes the following form:

```
set <EPISCRIPT_EXPANSION_VARIABLE> From_Configuration_Setting <CONFIG_
PATH>
```

For example:

```
set APPLOG From_Configuration_Setting Behavior/Debug/AppServerLogs
```

```
set ENCODING From_Configuration_Setting "External Interfaces/Meta/
DBEncodingScheme"
```

This can set “$$ENCODING” to “Single Byte (varchar and text).”

Setting the Value of DDLFILEPREFIX

The third form of the **set** command is a special case. It does not set the USE_FUNC_IDX expansion variable as the first form of command. Instead, it sets the DDLFILEPREFIX variable. While this form of the command must have either 0 or 1 as its argument, the value set in DDLFILEPREFIX is neither 0 or 1.

It takes the following form:

```
set USE_FUNC_IDX [0|1]
```

This sets the value of DDLFILEPREFIX to either “nf” (1 and the meta is an Oracle meta) or an empty string (0).

Predefined EpiScript Expansion Variables.

The following table describes the pre-defined EPISCRIPT_EXPANSION_VARIABLE variables.

Table 67: Scripting Macros

Macro	Description
$$CONFIGDIR	Expands to the path of the current configuration directory taken from the registry, command line, or file preferences form.
$$DATABASE_ PASSWORD$$DATABASE_ SERVER$$DATABASE_TYPE$$DATABASE_ USERNAME	Expand to the corresponding information taken from the connection of the EpiScript.
$$DDLFILEPREFIX	Expands to either “nf” or to an empty string. “nf”

Macro	Description
	Expands to “nf” if the meta is an Oracle meta and if the following are executed in the EpiScript: `set USE_FUNC_IDX 1` Empty string All other cases. See "Setting the Value of DDLFILEPREFIX" on page 401 for details.
$$ ENCODINGPREFIX	Expands to a value based on the configuration setting for DB_Encoding_Scheme. If configuration setting value is: ‘Single Byte (varchar and text)’ Results in an empty string results. (No prefix is added.) ‘Multi Byte (nvarchar and ntext)’ Results in a prefix of “n.”
$$ LOCALE	Expands to the current locale. The expanded value is initially taken from Admin Manager’s locale preference, but can be changed by the SET command. Use this prior to including script files that references locale-specific files Supplying DEFAULT as the NEW_VALUE restores the expansion of $$LOCALE to the value of the current Admin Manager default preference. This sets the locale for EpiScript expansion only. It does not declare it to be installed. Use install_locale to declare the locale as installed.
$$ LOGDIR	Expands to the name of the current logging directory. This refers to the date stamped subdirectory that is specific to this EpiScript execution. For example: `E:\Program Files\Infor\Campaign Management\10.1.0\JobLog\2011-10-14_09-59-39_886226_ommgr` and not `E:\Program Files\Infor\Campaign Management\10.1.0\JobLog`
$$NVARCHARPREFIX	Adds the N prefix to VARCHAR.

Calling Scripts in an Extraction Step

You can include script commands in system-call extraction steps. The following example illustrates the syntax for doing so. The $$ INSTANCEROOTDIR macro resolves to the pathname of the directory in which your EpiCenter instance is installed.

```
$$INSTANCEROOTDIR\bin\EpiMgrLaunch -f scriptname
```

Scripting Commands

The following sections describe the available scripting commands. Each section includes descriptions of commands that relate to the following categories:

For the most up-to-date information about these commands, refer to:

<installation_directory>\Infor\Infor Campaign Management\10.1.0\ConfigFiles\Docs\<locale>\Scripts

where <installation_directory> is the path to your Infor installation directory, and <locale> is the two-letter code for the language in which you wish to view the file.

SQL Script Assembly

The following table lists the available scripting commands that allow you to issue SQL against your EpiMeta.

Table 68: SQL Script Assembly Commands

Command	Description
check_log <FILENAME> [META | MART | OP]	Examines a SQL file for errors. Most database messages are treated as errors unless they are related to dropping objects that do not exist. If META, MART, or OP is specified, the check is based on the database type of that data store.

Command	Description
	Log checking is automatic for SQL executed in an Infor script.
execute_sql <FILENAME> [IGNORE_ERRORS] [META | MART | OP]	Opens the SQL interpreter tool from the database vendor (such as osql or sqlplus) and feeds it the SQL script specified with FILENAME. The syntax of the SQL must be acceptable to the database vendor. In practice, only the most simple SQL is acceptable to more than one vendor. If IGNORE_ERRORS is not specified, the output of the script is scanned for database-vendor-specific strings that are associated with most errors. If an error is found, the error is first scanned to determine whether it can be ignored (for example, if the error involves dropping a table that does not exist). If the error cannot be ignored, script execution halts. Error checking only occurs after the entire script has executed. It is up to the script designer to create scripts that halt when an error is found without executing subsequent SQL commands that causes damage. If META, MART, or OP is specified, the SQL executes against that database. META is the default.
execute_sql_with_extensions <FILENAME> [IGNORE_ERRORS] [META | MART | OP] [<NAME>=<VALUE> *]	Acts like the execute_sql command, but first passes the SQL script into the Infor SQL macro expander before sending the results to the database-vendor SQL interpreter tool. You can use the optional NAME/VALUE parameters to set private variables that exist in the script. See "Scripting Commands" on page 403 for information.
include <FILENAME>	Includes the contents of the SQL script specified with FILENAME inside the current script.
install_locale <locale_name>	Marks the named locale as having been installed. This command does nothing if the locale was already marked as installed.
set <EPISCRIPT_EXPANSION_VARIABLE> <NEW_VALUE>	Changes the expanded value of Admin Manager variable specified by EPISCRIPT_EXPANSION_VARIABLE to NEW_VALUE. Use DEFAULT as the NEW_VALUE parameter to restore the expansion of the variable to the value of the current Admin Manager preference.

Command	Description
	Currently this command can only change the expansion of $$LOCALE.

SQL script designers must also include any script directives that are required to make the database-vendor script interpreter work correctly. Oracle scripts, for example, usually need a spool command and sets of values such as `echo`, `trim`, and `define`. Oracle scripts must also include an `exit` command, otherwise control is never returned to the Infor script processor.

Private Variables in SQL Scripts

SQL scripts executed by Admin Manager's script interface can include private variables with the either of the following lines:

```
EpiMgr BOOLARG <argname> <prompt text>
```

```
EpiMgr STRINGARG <argname> <prompt text>
```

argname is the name of the private variable in your SQL script, and prompt text is the prompt for the variable's value that is displayed to users when they run the script from Admin Manager's Windows interface. (See "Executing Scripts within Admin Manager" on page 396.)

For example,

```
-- EpiMgr STRINGARG PHYSICALPROPS What are the physical properties?
```

causes `$$[PHYSICALPROPS]` to expand before script processing. If the script is initiated from a tools menu in Admin Manager, the user is prompted for the value of the arguments. If the script is called with the `execute_script_with_extensions` command, the value is specified in a `name=value` pair. For example,

```
execute_script_with_extensions createtables.sql "PHYSICALPROPS=Meta
Space Index 5"
```

Embedded Commands in SQL Scripts

SQL scripts executed by Admin Manager's script interface can also include the commands listed in Table.

Table 69: Embedded Commands in SQL Scripts

Command	Description
-- EpiMgr EPIMETA-- EpiMgr EPIMART-- EpiMgr EPIOP	These commands cause the SQL to be executed against the named database. This directive overrides the META, MART, or OP specification in the episcript command.
-- EpiMgr REFRESH	Causes Admin Manager to suggest that the user must refresh the dialog box. This command is only useful when the script is run within Admin Manager's user interface.
-- EpiMgr SHOW	Causes Admin Manager to display the log file. This is useful when the SQL output is a report for the user to examine. This command is only useful when the script is run within Admin Manager's user interface.
-- EpiMgr WARNING <text>-- EpiMgr WARN <text>	Causes Admin Manager to present the text as a pop-up warning before the script executes. This command is only useful when the script is run within Admin Manager's user interface.
-- EpiMgr DESCRIPTION <text>-- EpiMgr DESC <text>	Describes the script. If multiple description lines exist the text is appended. Currently descriptions are not shown.

SQL Script Examples

- To call a custom script installed outside of Infor:

```
execute_sql "D:/My Documents/Mysql.sql"
```

- To perform EpiOp operations:

```
execute_sql EpiOp/Oracle/EpiOp_drop.sql IGNORE_ERRORS OP

execute_sql EpiOp/$$DATABASE_TYPE/EpiOp_persistent_DDL.sql OP
```

- To call a tools script and supply values for the macros declared in its header:

```

```

```
execute_sql_with_extensions "UpdateScripts/Topic/Configure/Unbind_User_
Web_pages.SQL" 'TOPIC_NAME=Sales - Concluded'
```

- To call a script relative to the `ConfigFiles` dir:

```
execute_sql_with_extensions "EpiMeta\generic\Core\Static_Trans\pre_
scan.SQL"
```

- To execute generated SQL, first call a script with header macros, then pass its output as another script:

```
execute_sql_with_extensions "EpiMeta\generic\Core\Static_Trans\gen_
scan.sql" IGNORE_ERRORS NEWLYINITED_DBNAME=testInit600_epimeta"execute_
sql_with_extensions "$$LOGDIR\gen_scan.sql.log"
```

Database Connections

The following table lists the available scripting commands that allow you to connect to an EpiMeta database.

Table 70: Database Connection Commands

Command	Description
close_meta	Closes the connection to the EpiMeta database that was last opened.
connect_meta <VENDOR> <SERVER> <DATABASE> [<USERNAME> <PASSWORD>]	Connects to the database specified in the DATABASE parameter. Use this command if EpiMeta tables already exist in the database. The VENDOR parameter is case-sensitive and must be `SQLServer`, `Oracle`, or `DB2`. For Oracle connections, use the same value for DATABASE and USERNAME.
connect_meta_for_init <VENDOR> <SERVER> <DATABASE> [<USERNAME> <PASSWORD>]	Connects to the database specified in the DATABASE parameter. Use this command for uninitialized databases. The VENDOR parameter is case-sensitive and must be `SQLServer`, `Oracle`, or `DB2`. For Oracle connections, use the same value for DATABASE and USERNAME.

Command	Description
connect_meta_instance <INSTANCE_NAME>	Connects to an EpiCenter instance that is already recognized by Admin Manager.

Data Store Settings

The following table lists the available scripting commands that allow you to alter data store settings for the EpiMart and EpiOp databases. EpiMeta data store settings can only be set with the `connect_meta` command prior to a database initialization.

Table 71: Data Store Settings Commands

Command	Description
set_datastore<DATASTORENAME> <VENDOR> <SERVER> <DBNAME> [<USER> [<PWD> [<PORT>]]]	Alters the settings for a non-ODBC data store to the values that are specified in the command. PORT is ignored with SQL Server databases.
set_epimart <VENDOR> <SERVER> <DBNAME> [<USER> [<PWD> [<PORT> [NO_TEST]]]]	Alters the EpiMart data store setting to use the values that are specified in the command. If NO_TEST is not supplied the database connection is opened to verify that the connection information is valid. If this test fails the script aborts.
set_epiop <VENDOR> <SERVER> <DBNAME> [<USER> [<PWD> [<PORT> [NO_TEST]]]]	Alters the EpiOp data store setting to use the values that are specified in the command. If NO_TEST is not supplied the database connection is opened to verify that the connection information is valid. If this test fails the script aborts.
set_physical_properties [META | MART | OP] | [<&&2table_properties> <&&3index_properties>]	Sets the value of the &&2 and &&3 variables in Oracle scripts that have “set define on” specified: &&2 is generally reserved for the physical properties of create table commands. &&3 is generally reserved for the physical properties of create index commands. If this command is called with no arguments, the current physical properties are displayed. If this command is called with META, MART, or OP as the sole argument, table properties are set to the expansion of: `$$TABLESPACE_CLAUSE[METADATA~,~ ] $$TABLE_PROPERTIES_OVERRIDE [$$ QUERY_DOP[METADATA]~,~$$EXTENT`

Command	Description

	`[METADATA]~,~$$PCTFREE[METADATA]~,~$$PCTUSED[METADATA]]`
	
	and index properties are set to the expansion of:
	
	`$$TABLESPACE_CLAUSE[METAIDX~,~ ] $$TABLE_PROPERTIES_OVERRIDE [$$QUERY_DOP[METAIDX]~,~$$EXTENT [METADATA]~,~$$PCTFREE[METADATA]~,~$$PCTUSED[METAIDX]]`
	
	where METADATA and METAIDX are replaced with the appropriate values.
	
	This command returns an error if used on an uninitialized EpiCenter.

Schema Generation

The following table lists the available scripting commands that allow you to perform operations related to schema generation.

There is no command to initialize an EpiMeta database because initialization occurs when a script from the following directory is included in a script:

```
<install_directory>\ConfigFiles\Configurations\init
```

For example, the following line initializes an EpiMeta database even if it has already been initialized:

```
include Configurations\init\reinit.epi
```

Table 72: Schema Generation Commands

Command	Description
`date_populate`	Populates the date dimension. See "Generating the Schema" on page 161 for information.
`finish_initialization`	Used at the end of initialization, this command sets: the build number configuration setting.

Command	Description
	the Meta, Op, and Mart data stores to the correct database vendor. the maximum number of columns and parallel degree physical object properties. default object properties.
`finish_initialization_complete`	Marks metadata after initialization completes successfully. This command typically runs in an initialization script after the finish_initialization command.
`GENERATESCHEMA [TRIALRUN] [FORCEREBUILD] [REBUILDNECESSARY] [ADAPTTABLE] [BUILDBACKFEED] [RANDPERM]`	Generates your schema according to the settings that you include. See "Generating the Schema" on page 161 for information about each option: [TRIALRUN] or [T] - trial run. [FORCEREBUILD] or [FB] - force rebuild. [REBUILDNECESSARY] or [RN] - rebuild when necessary. [ADAPTTABLE] or [AT] - adapt table. [BUILDBACKFEED] or [BBF] - build backfeed tables. [RANDPERM] or [RND] - build sampling tables.
`init_epiop`	Initializes an EpiOp database using the parameters that are currently set.
`initialize_configuration`	Sets the CurrentState configuration settings to values that are consistent with an ungenerated schema. Sets the product's version, build, and initialization script name. Sets DBEncodingScheme to 'Single Byte (varchar and text)' This command must be executed before importing values that override the parameters above.
`populate_sample`	Builds the dimension sampling table. See "Generating the Schema" on page 161 for information.
`purge <DATASTORENAME>`	Removes all tables and views that are not required in your EpiCenter from the specified data store.
`verify_mart`	(Boolean) Returns True if the connection to the EpiMart is good and the mart is initialized; returns False otherwise.
`verify_op`	(Boolean) Returns true if the connection to the EpiOp is good and the EpiOp is initialized; returns False otherwise.

Dimension Tables

The following table lists the available scripting commands that allow you to control dimension tables.

Table 73: Dimension Table Commands

Command	Description
`dim_define <DIMNAME> [KEYTYPE=<TYPE> SSKEYTYPE=<TYPE> SAMPLE=<INVLOG10> PARTS=<NUMPARTS> REVISIONPCT=<PCT> NEWNAME=<DIMENSIONNAME>]`	Updates an existing base dimension, setting many of the properties on the base dimension form. This function cannot add a new dimension. Valid types are: UNMAPPED | MAPPED VERSIONED | NOT_VERSIONED MEMORY_LOOKUP | NOT_MEMORY_LOOKUP LIST_PRODUCING | NOT_LIST_PRODUCING BACKFED | NOT_BACKFED SEEDED | NOT_SEEDED HAS_INFERRED_RESPONSES | HAS_NO_INFERRED_RESPONSES EM_ENABLED | NOT_EM_ENABLED RT_ENABLED | NOT_RT_ENABLED FUSION_ENABLED | NOT_FUSION_ENABLED
`dim_delete <DIMNAME>`	Deletes a base dimension.
`dim_col_define <DIMNAME> <DIMCOLNAME> [TYPE=<TYPE> NEWNAME=<NEWCOLNAME> DEFAULT=<DEFAULTVALUE>]`	Adds or updates a dimension column with the given properties. Valid types are: INDEX | NOT_INDEX QUERYABLE | NOT_QUERYABLE INTEGER_MAP | NOT_INTEGER_MAP LOW_CARDINALITY | NOT_LOW_CARDINALITY
`dim_col_delete <DIMNAME> <DIMCOLNAME>`	Delete the given dimension column in the specified dimension.

Fact Tables

The following table lists the available scripting commands that allow you to control fact tables.

Table 74: Fact Table Commands

Command	Description
`fact_define <FACTNAME> [SSKEYTYPE=<TYPE> ROLLOFFGRAIN=<ROLLOFFGRAIN> ROLLOFFUNITS=<NUMUNITS>NEWNAME=<FACTNAME>]`	Updates an existing fact table, setting its properties. This operation cannot be used to create a new fact table. Valid types are: TRANS_AND_STATE | NO_TRANS_AND_STATE
`fact_delete <FACTNAME>`	Deletes a fact table.
`fact_col_define <FACTNAME> <FACTCOLNAME> [TYPE=<TYPE> NEWNAME=<NAME>]`	Adds a new fact column to a fact table or updates an existing fact column.
`fact_col_delete <FACTNAME> <FACTCOLNAME>`	Deletes a fact column from a fact table.
`fact_dim_role_add <FACTNAME> <DIMNAME>|<DEGENNAME>`	Adds a dimension role (either regular or degenerate) to a fact table. **Note:** you do not need to rebuild the fact table after adding a dimension role.
`fact_dim_role_delete <FACTNAME> <DIMNAME>|<DEGENNAME>`	Removes a dimension role (either regular or degenerate) from a fact table. **Note:** you do not need to rebuild the fact table after removing a dimension role.
`fact_agg_enable <FACTNAME> <AGGNUMBER> <ENABLED>{0|1}`	Enables or disables a fact aggregate instruction (by number) for the given fact table. The **<ENABLED>** parameter must be either 0 or 1.

Job Control

The following table lists the available scripting commands that allow you to control extraction jobs.

Table 75: Job Control Commands

Command	Description
`cancel_job`	Cancels a job that has already been rolled back. You cannot commit a job after you have cancelled it.
`commit_job`	Switches to the set of mirrored tables that was made active by the most recent extraction.

Command	Description
`execute_job <JOBNAME> [MaxSelects] [ProcessAttach] [ProcessKill] [MailHost] [MailAdmin] [MailSuccess] [MailFailure]`	Invokes epichannel with the extraction job specified by JOBNAME. The arguments correspond to the options in the Execute Job dialog box. See "Executing a Job" on page 312.
`rollback_job`	Switches the set of mirrored tables that was active before the most recent extraction. Backfeed tables are not affected.

Topics

The following table lists the available scripting commands that allow you to manipulate topics.

Table 76: Topic Commands

Command	Description
`bind_topic [TOPIC_NAME] [ALLNODES | UNBOUND] [CONTEXT]`	Binds the nodes in a given topic to a specified context. Use ALLNODES | UNBOUND to control whether all navigation nodes in the topic are bound or only those nodes that are currently unbound.
`update_topic <TOPIC_NAME> <TEMPLATE_TOPIC_NAME|DEFAULT> [<option>=<value>]...`	Performs a topic update using either the named template topic or the template that is already associated with the topic as the pattern. The options correspond to the checkboxes in the Topic Upgrade dialog box. Valid options are: TITLE_PROMPT_DESCRIPTION_CHANGES (default value=1) NEW_NAV_NODES (default value=1) RESET_ENTRY_NODE (default value=1) COPY_WEB_BINDINGS (default value=1) COPY_ONLY_NEW_WEB_BINDINGS (default value=0) RESET_NAV_NODE_POSITIONS (default value=1) DELETE_PROTECTED_LINKS (default value=0) DELETE_LINKS_TO_INTERNAL (default value=1) DELETE_LINKS_TO_EXTERNAL (default value=0) NEW_NAV_LINKS (default value=1) Valid values are 0 (for false) or 1 (for true). See the Topic Implementation Guide for details about these options.

Command	Description
`webpage_bind <WEB_PAGE_NAME> [DIMEN-SION<1>] ... [DIMENSION<N>]`	Disassociates all dimensions from the named Web page and then associates the named dimension(s) to that page. If no dimensions are included, the Web page is no longer be associated with any dimension.

Users and Groups

The following table lists the available scripting commands that allow you to edit users and groups.

Table 77: User and Group Editing Commands

Command	Description
`add_group_column_security <GROUPNAME> <DIMROLE> <DIMCOLUMN> [<VALUE>*]`	Restricts the group specified by GROUPNAME to a subset of the data in your EpiMart. Use DIMROLE and DIMCOLUMN to specify the dimension column that you wish to restrict. Enter one or more VALUE entries to specify the values that members of the group can view. For example, selecting Date.fiscal_year as the dimension role and column and the values 1997 and 1998 causes all reports to be filtered with these values. If you do not wish to restrict access to a column, do not specify any access entries for that column.
`add_group_member <USERNAME> <GROUP-NAME> [PRIMARY]`	Adds the user specified by USERNAME to the group specified by GROUPNAME. Use the PRIMARY option to specify that this is the user's primary group. The restrictions set for the primary group apply to all queries initiated by users in that group. A user also inherits default permissions from his or her primary group.
`add_user_column_security <USERNAME> <DIMROLE> <DIMCOLUMN> [<VALUE>*]`	Restricts the user specified by USERNAME to a subset of the data in your EpiMart. Use DIMROLE and DIMCOLUMN to specify the dimension column that you wish to restrict. Enter one or more VALUE entries to specify the values that the user can view. For example, selecting Date.fiscal_year as the dimension role and column and the values 1997 and 1998 causes all reports to be filtered with these values. If you do not wish to restrict access to a column, do not specify any access entries for that column.

Command	Description
`clear_group_membership <GROUPNAME>`	Removes all members from the group specified by GROUPNAME. This action leaves the group empty.
`clear_user_membership <USERNAME> [PRIMARY_ONLY]`	Removes the user specified by USERNAME from all groups. Specify PRIMARY_ONLY if you wish to clear the user's primary group setting without removing group memberships.
`group_define <GROUPNAME> [SYNCHRONIZE NOT_SYNCHRONIZE ADMINISTRATOR NOT_ADMINISTRATOR]`	Creates a new group or updates an existing group. Use the SYNCHRONIZE option to ensure that the group listing in metadata matches the group listing in your Windows or LDAP domain. If you do not wish to synchronize the group, specify NOT_SYNCHRONIZE. Use the ADMINISTRATOR option if members of this group have administrative rights. An administrator can see all web pages and reports in the system, but cannot modify special folders, such as the Public folder. Use the NOT_ADMINISTRATOR option if members of this group do not have administrative rights.
`group_delete <GROUPNAME>`	Deletes the group specified by GROUPNAME from your metadata. This action does not delete the users who are members of the group.
`user_delete <USERNAME>`	Deletes the user specified by USERNAME from your metadata.
`user_insert <USERNAME> <FULLNAME>`	Creates a new user.

Import and Export

The following table lists the available scripting commands that allow you to import and export your metadata to a `.mdb` file. Note that metadata export works by first creating a list of objects to be exported and then performing the export with the `export_metadata` command. `import_metadata <FILENAME>`

Table 78: Import and Export Commands

Command	Description
`add_revision_history <text_description>`	Adds text_description to the EpiCenter's revision history (accessible in Admin Manager from the EpiCenter menu).

Command	Description
	For example, instead of including only a series of exports, imports, and jobs, this command can cause the revision history to also include text like, “Moved from test to production.”
`export_metadata <EXPORTFILENAME> [SUPPRESS_CAPILLARY_EXPORT] [DROP_EDIT_FLAG] [<DESCRIPTION>]`	Creates an export file based on the existing list of objects that must be exported. This list is edited with the other export commands described in this table. Some exported objects contain “capillary” references to other objects. Capillary references occur when one object is required before an object can be loaded but the former object is not part of the latter object. For example, the users of a report are capillary dependencies of the report. The users must exist when the report is imported, but the users are not themselves defined by the report. A non-capillary export are navigation nodes within a topic. A navigation node cannot exist outside of its topic. When SUPPRESS_CAPILLARY_EXPORT is supplied, objects that are found capillary references are not exported unless they are explicitly included in an export selection. For example, it is possible to use this to export the reports and not also export the users of the reports. When DROP_EDIT_FLAG is supplied, the edit_flag column is omitted from the export file. This option clears the edit flag settings for built-in objects such as the Date dimesion.
`export_metadata_add_sequence <SEQUENCENAME> <WHERECLAUSE> [RECURSIVE]`	Adds objects from the given export sequence that match the WHERECLAUSE to the list of objects that must be exported. Export sequence names and their contents can be found in Admin Manager by selecting Tools > Miscellaneous > Describe Export.

Command	Description
	If RECURSIVE is supplied, objects are exported recursively.For example, if you choose to export a folder, all folders contained within the specified folder are also exported. Use caution when using this operation. Built-in objects that must not be exported can possibly be exported if an incorrect WHERECLAUSE is provided.
`export_metadata_add_set <SETNAME>`	Adds the default selection from the named export set to the list of objects that must be exported. (Export sets are visible in the Export dialog box.)
`export_metadata_clear`	Empties the list of objects that must be exported.
`export_metadata_remove_sequence <SEQUENCENAME>`	Removes an export sequence from the list of objects that must be exported. Export sequence names and their contents can be found in Admin Manager by selecting Tools > Miscellaneous > Describe Export.
`import_metadata <FILENAME>[ALWAYS_REPLACE NEVER_REPLACECONTINUE_AFTER_ERRORMAX_MESSAGES=<N>SCHEMA_MERG`EDELETE_NOT_IMPORTED-NO_DUP_CHECKINGNON_VERBOSE_LOGGING-DO_NOT_LOG_OVERWRITES-DO_NOT_LOG_BUILT_IN_SKIPDO_NOT_LOG_PARENT_NOT_IMPORTEDDO_NOT_LOG_MISSING_OPTIONAL_REFERENCESALLOW_UPGRADE]	Imports the .mdb file specified by FILENAME into your EpiCenter. ALWAYS_REPLACE NEVER_REPLACE. Specifies how to handle objects whose names clash with existing names. CONTINUE_AFTER_ERROR. Specifies whether to continue importing if an error is encountered. MAX_MESSAGES. Specifies the maximum number of import log messages to tolerate before failure. SCHEMA_MERGE. Specifies whether to merge the contents of dimensions or facts when a schema object with the same name as an existing entry is imported. See "Schema Merging" on page 437 for information. DELETE_NOT_IMPORTED. Specifies whether to delete objects from certain import categories (such as attributes) if those objects are not mentioned in the import file (destructive import). This

Command	Description
	option is useful for development to production rollout to purge stale entries in the production metadata. NO_DUP_CHECKING. Specifies whether rows are inserted without checking to see if they already exist in the meta. NON_VERBOSE_LOGGING. Specifies that summary information must only be displayed in the import log message. DO_NOT_LOG_*. Specifies whether to suppress certain import messages. ALLOW_UPGRADE. Specifies whether the import file can be the result of an upgrade operation.
`import_template <FILENAME>`	Imports a text file that contains the semantic type definitions.

TUM

The following table lists scripting commands that perform translation-related actions. See “Multi-Lingual Epicenters” in chapter 4 of the *Infor Campaign Management Topic Implementation Guide* for details of the TUMs utility.

The examples in table use the following defined string arguments:

- $$PROJECT_META_DIR

  The name of the directory in which the project translation files are stored. A typical value is $$CONFIGDIR\Metadata.

- $$TRANSSET

  The name given to the set of strings extracted from metadata. A typical value is Custom_AS.

Table 79: TUM Commands

Command	Description
`TUM_Connect [MDBFILENAME]`	Sets the TUM file upon which all the commands operate. This command opens a connection to the

Command	Description
	file so the file is locked for some purposes until TUM_close is run. For example: `TUM_connect "$$PROJECT_META_DIR/$$ LOCALE/TUM/$$TRANSSET_MTUM.mdb"`
`TUM_close`	Closes the TUM file opened by the TUM_connect command.
`TUM_extract`	Performs the equivalent action as that initiated by the Tools > Internationalization > Translate User Label menu command.

`TUM_set <PARAMETER> <VALUE>`	Sets the named parameter to the indicated value. Valid parameters include:
	Transet - Sets the name of the set of extracted strings.
	
	`TUM_set TransSet $$TRANSSET`
	
	AS_CONFIG_ROOT - Sets the location of the installed templates.
	
	`TUM_set AS_CONFIG_ROOT "$$CON- FIGDIR"`
	
	Project_Meta_Dir - Sets the location of the project translation files.
	
	`TUM_SET Project_Meta_Dir "$$ PROJECT_META_DIR"`
	
	For a complete list of valid set parameters, see the comments in the “Translate_User_Labels.epi” EpiScript in the following directory:
	
	`<install_directory>\ConfigFiles\ UpdateScripts\MainScreen\I18n`

Command	Description

`TUM_set_locales <Locale>`	Sets the locales used by the TUM utility. The following example sets them automatically based on the locales which are already installed with the current locale being the source locale.
	`TUM_set_locales $$LOCALE`
	Alternatively, you can hard code the set of locales by doing the following:
	Set the source locale of the translated strings:
	`TUM_set Master_Locale en`
	`TUM_set locale en`
	Set the available locales (including the master locale)
	`TUM_set locales "de en es fr ja ko pt"`

`TUM_reset_log`	Re-sets the log file length to zero.

Locales

The following table lists scripting commands that perform locale-related actions.

Table 80: Locale Commands

Command	Description
`include_for_other_locales <FILENAME>`	Similar to the include command ("Table 68: SQL Script Assembly Commands" on page 403) in that it includes the contents of the SQL script specified with FILENAME inside the current script. However, for every locale installed in the metadata (except for the current locale), it switches to that locale and then runs the script. It skips the following locales:

Command	Description
	any locale for which the appropriate supplemental files are not installed any locale that require an incompatible code page (For example, it skips Japanese if your machine is currently set up for English.)
`set_translation_role <TRANSLATION_ROLE_NAME> <LOCALE_NAME>`	Sets the translation role to use for the named locale. Scrutiny errors results if the locale is not installed. Translation roles define the locale used for a set of messages. Valid roles are: Frontend The Infor Campaign Management Server front end strings. Multiple locales sets as a Frontend locale. Logging Infor Campaign Management logging strings, including those for EpiChannel and Infor Campaign Management Server. This role causes the deletion of any prior settings for this role. Supplying DEFAULT as the LOCALE_NAME causes EpiScript to use Admin Manager's default metadata locale.

Statistics

The following table lists scripting commands that involve statistics.

Table 81: Statistics Commands

Command	Description
`enable_query_stats [IGNORE_ERRORS]`	Enables an Oracle meta to select from the EpiOp for query statistics. If the optional IGNORE_ERRORS argument is given, the command does not generate an error if it is used with an EpiCenter for which query logs cannot be enabled.
`refresh_epimeta_stats`	Refreshes database statistics on EpiMeta metadata objects.
`refresh_epiop_meta_stats`	Refreshes database statistics on EpiOp metadata objects.

Miscellaneous

The following table lists additional scripting commands which do not readily fall into one of the previous categories.

Table 82: Miscellaneous Commands

Command	Description
`discard_saved_lists`	Deletes all materialized lists from your EpiCenter. Non-materialized list definitions are saved.
`duplicate <EXPORT__NAME><OB-JECT_NAME><NEW_OBJECT_NAME>`	Copies the object. The new object is located in the same navigation folder as the original. **Note:** If you copy a built-in object, the new object is editable.
`exitunconditional_exit`	Halts the current script with a normal exit code. If the current script was called with another script (with an “include” command), the calling EpiScript continues. EpiScript does not execute the exit statement if it is inside an if block that is not being executed. However, EpiScript always executes the unconditional_exit statement.
`print <text>`	Causes the command line to print the string specified by text to verbose logs without including a timestamp. This command allows you to add explanations or formatting to the logfiles.
`process_query_logs [<THRESHOLD> RELOAD_ALL]`	Stores query statistics in your metadata so that you can optimize your aggregate instructions. Use THRESHOLD to specify the number of queries that must be made against a particular attribute or group of attributes before those queries are included in the statistics. Use RELOAD_ALL to specify that all queries that are specified since your EpiCenter is first instantiated must be counted. Otherwise, only those queries that are specified since the last set of queries are processed are counted.

Command	Description
`redirect_verbose <NEW_VERBOSE_LOG_NAME>`	Causes logging of detailed operations to occur only in the named script file. Other logging occurs in both the current log file and in the detailed log file. If you wish the detailed log file to be saved in the logging subdirectory, you can use a command such as: `redirect_verbose $$LOGDIR\Detailed.log`
`set_macro <MACRONAME> <VALUE> <STORETYPE>`	Sets the macro specified by MACRONAME to the given VALUE. Use STORETYPE to specify the data store.

Mart Management Commands

You can use mart management commands to perform common Infor Campaign Management Server maintenance operations using a command-line utility (AdminClient) or an SQL Server stored procedure (xp_epiappserver).

- "The Admin Client Utility" on page 423 describes AdminClient command execution.
- "The xp_epiappserver Stored Procedure" on page 424 describes xp_epiappserver command execution.
- "The Mart Management Commands" on page 425 describes the mart management commands themselves.

The Admin Client Utility

You use the AdminClient utility to issue mart-management commands interactively. It can be found in <instance-dir>\AP\bin (for example, `C:\Program Files\Infor\Campaign Management\10.1.0\Web\<instance-name>\AP\bin`).

AdminClient returns zero when a command is successful. If a command fails, a non-zero error code and error message appear at the console.

AdminClient	-url <Infor Campaign Management_Server_url> -user <user_name> -password <password> <mart_management_command [argument]>
-url <Infor_Campaign_Management_Server_url>	The <Infor_Campaign_Management_Server_url> argument provides the URL to Infor Campaign Management Server. If you omit a protocol, HTTP

	is presumed. If you omit a port identifier, 80 is presumed.
-user <user_name>	The <user_name> argument provides the user name of an individual with administrator privileges.
-password <password>	The <password> argument is the user's password.
<mart_management_command [argument]>	The <mart_management_command> argument is one of the commands described under "Mart Management Commands" on page 423. [argument] is a value passed into those commands that accept an argument.

Sample Usage

```
AdminClient -url http://srvr.cmpny.com:80/scripts/inst/epiphany.dll -
user admin -password admin refresh
```

The xp_epiappserver Stored Procedure

This stored procedure allows you to programmatically submit mart management commands. If xp_epiappserver encounters an error, a non-zero value is returned. "Table 83: Return Codes for xp_epiappserver" on page 425 describes the return codes and values.

xp_epiappserver	'Infor Campaign Management_Server_url', 'meta_DB_name', 'command_name' [,'argument']
Ensure that you enclose each parameter value in single quotation marks and separate parameters using a comma.	
'Infor Campaign Management_Server_url'	The 'Infor Campaign Management_Server_url' parameter provides the URL to the Infor Campaign Management Server. If you omit a protocol, HTTP is presumed. If you omit a port identifier, 80 is presumed.
'meta_DB_name'	The 'meta_DB_name' parameter identifies a metadata database.
'command_name' [,'argument']	The 'command_name' parameter is one of the commands described under "The Mart Management Commands" on page 425. The 'argument' parameter is a value passed as an argument to commands that accept an argument.

Installation

To install the xp_epiappserver stored procedure, perform the following steps.

Installing the xp_epiappserver Stored Procedure

Copy the library AppServerStoredProcedure.dll from the Resources directory of your Infor Campaign Management installation disk to the Bin subdirectory of your SQL Server installation directory.

1 Invoke the sp_addextendedproc stored procedure in the master database of your SQL Server installation to add the xp_epiappserver stored procedure. Do this by issuing the following command against the master database in Query Analyzer:

```
sp_addextendedproc 'xp_epiappserver', 'AppServerStoredProcedure.dll'
```

2 Configure an ODBC DSN named `LocalServer` for use by xp_epiappserver. If you are using NT authentication, the DSN must be configured to use integrated NT authentication.

3 Run SQL Server under a domain user account, not under the default Local System account. Give the domain user account permission to access the web server used by the Infor Campaign Management Server, using the IIS configuration tool in the control panel.

Sample Usage

```
declare @res INT
EXEC @res = master..xp_epiappserver
'http://machine/scripts/instance/epiphany.dll', 'myDB', 'refresh'
print @res‘
```

The Mart Management Commands

Table 83: Return Codes for xp_epiappserver

Return Code		Description
0	SUCCESS.	The stored procedure executed successfully.
1	UNKNOWN_ERROR	The command can not be executed because an unknown error occurred.

Return Code		Description
2	BAD_ARGS	The arguments provided are invalid. Ensure that you include the correct number of arguments and that they are all strings.
3	ARG_READING_FAILURE	The arguments can not be read. Ensure that all arguments are strings and that arguments are delimited using single quotation marks.
4	DATABASE_WRITE_ERROR	There is a problem in writing to the database. This error can occur because the LocalServer DSN is not configured properly or because the database argument is incorrect.
5	APPSERVER_NOT_AVAILABLE	A connection to the specified Infor Campaign Management Server can not be specified.
6	UNKNOWN_WEB_PAGE	An unknown web page returns. Ensure the URL you specify is correct.
7	INVALID_CONTENT_TYPE	The web server returned an image or some other content type rather than text/xml or text/html. Ensure the URL you submit is correct.
10	WINDOWS_ERROR	A Windows error occurred. The number of errors in this category is large. Refer to the printed message for specific information about the nature of the error that occurred.
11	HTTP_SERVER_NOT_FOUND	The Infor Campaign Management Server host identified in the URL is not available.
13	Client Authentication	The user executing the xp_epiappserver stored procedure is not successfully authenticated by the Infor Campaign Management Server.
1nnn	HTTP_SERVER_ERROR_BASE	The HTTP Server returned a status code, identified by nnn. Com-

Return Code		Description
		mon examples are 1401and 1500: 1401 indicates an HTTP status of 401, access denied. Ensure that the account under which SQL Server is running has access to the target web page. 1500 indicates an HTTP status of 500, server internal error. An invalid URL can cause this error to occur. Use the following URL to review more information about status codes: http://www.webpartner.com/html/AlertsandErrors.htm.
1000 - 2000	Infor Campaign Management Server error code	An error code returned by the Infor Campaign Management Server. An error in this range occurs when a request reaches the Infor Campaign Management Server, but the request cannot be processed.

You can use the AdminClient and xp_epiappserver utilities to run the mart-management commands described in this section and summarized in Table .

Table 84: Mart Management Commands

Mart Management Command	Description
freeze <MaxWaitTime>	Temporarily suspends operations. Use freeze before backing up or restoring metadata or the EpiOp database. Use the refresh and unfreeze commands following these operations to specify the Infor Campaign Management Server available again. < MaxWaitTime > is an optional argument that specifies the number of seconds to wait for currently running queries to finish before killing them and dropping all database connections. If you pass in -1 for this option, the Infor Campaign Management Server waits indefinitely for currently-running queries to finish before freezing. The freeze command can throw the following exception:

Mart Management Command	Description
	InvalidArgument The MaxWaitTime value is not a non-negative integer or -1.
generate_semantic	Regenerates custom semantics. The generate_semantic command takes a single argument which can be either of the following: a semantic name *All_Custom* (with the asterisks)
martCheckpoint	Flushes lists from the EpiMart to the EpiOp database. For example, you can issue this command in preparation for backing up the EpiOp database. The martCheckpoint command can throw the following exception: MartCheckpointException The Infor Campaign Management Server encountered an error while flushing list tables to the EpiOp database.
refresh	Directs the Infor Campaign Management Server to reread all metadata. The refresh command can throw the following exception: RefreshException An error occurred while refreshing the metadata.
unfreeze <performRefresh?>	Reactivates suspended Infor Campaign Management Server operations. Connections between the Infor Campaign Management Server and the EpiMeta and EpiOp databases are restored. If <performRefresh?> is true, a refresh is automatically performed after the unfreeze. If no boolean value is specified, the default is false.

Exporting and Importing Metadata

15

This chapter describes facilities in Admin Manager that allows you to export metadata from one Infor Campaign Management instance and import that metadata into another instance. These facilities allow you to:

- Back up and restore the definition of a data mart and associated applications.
- Migrate selected updates from a test system to a production system.
- Migrate metadata from previous Infor Campaign Management releases into the current release.
- Export metadata based on modification date.

You can migrate an entire EpiMeta database, specific metadata objects, or categories of objects that are stored in metadata, so long as the language and locale of the exported metadata matches the language and locale of the data mart in which that metadata is to be imported. Admin Manager automatically converts characters within the same language from one code page to another during an import operation .

When you register two data marts in the same Admin Manager window, you can use the drag-and-drop interface to migrate items or folders from one EpiMeta database to another. When you drag an item or folder from one EpiCenter folder to another, Admin Manager initiates the appropriate export and import operations to transfer the metadata that you have chosen. If you pick a folder to drag and drop, Admin Manager migrates all of the subfolders and objects within that folder.

Exporting Metadata

It is not always feasible to use drag and drop to migrate metadata between data marts, particularly when you are upgrading from an earlier release. Admin Manager provides an **Export** option in the **EpiCenter** menu that you can use to export metadata from one data mart to a file. You can then use the Import option in the **EpiCenter** menu for another data mart to import metadata from that file. The export file takes the form of a Microsoft Access database (.mdb) file.

To ensure that primary keys are not corrupted by an import operation, the export file uses a database-independent representation of metadata. See "Export File Format" on page 439 for a description of this representation format.

The export operation for a particular object might include additional antecedent and descendent objects that are required to recreate a consistent image of the object that you have chosen. For example, an attribute is exported with its filter groups and elements. The import operation resolves primary keys and integrity constraints as it incorporates metadata into the data mart.

Built-In Objects and the Edit Flag

The export process does not include unmodified built-in objects that are provided by Infor. Most built-in objects are created when you initialize your EpiCenter. Examples of such objects include the Date and Campaign dimensions, extraction job templates, and topic templates. Other built-in objects are introduced by user action, including derived facts from list-producing dimensions, and Infor Email Marketing built-in columns that are added to a Email Marketing-enabled dimension.

All built-in objects include an `edit` flag that is initially set to false. This value indicates that no changes to the object and that it must not be included in export operations. If you modify a built-in object in any way, the `edit flag` for that object is set to true and the object is included in future export operations. For example, if you export the **User Defined Metadata** export set, the resulting .mdb file includes all edited built-in objects and custom objects, while excluding the “runtime” metadata that is generated by the Infor Campaign Management system to describe the current status of tables and indexes in the EpiMart database.

Note: Edit flags are only used for built-in objects, and have no effect on the export of custom objects that you define.

Before you import a .mdb file into another EpiCenter, you must ensure that the appropriate non-edited built-in objects are present in the EpiCenter prior to import. If these objects do not exist, `Import parent reference not found` errors occur. For example, before you import an Email Marketing-enabled .mdb file, you must initialize the target EpiCenter with the Email Marketing option so that the Email Marketing-related built-in objects are present.

Note: If you've exported from a meta that has Marketing Resource Management enabled, then you need to import into a meta that has Marketing Resource Management enabled.

When exporting, you can choose the **Clear Edit Flags** option to specify that all built-in metadata with the edit flag set to true must be exported with the flag set to false in the .mdb file. As a result:

- When the .mdb file is imported to a new EpiCenter, rows in the .mdb with cleared flags are imported if the Preserve Edits import option are not selected.
- Future exports of this metadata (after it is imported into a new EpiCenter) do not include the rows with cleared flags, unless changes are made to the rows in the interim (thereby changing the edit flag back to True). These exports can fail to import due to missing object references unless this export file is imported first.

Exporting Metadata by Date

Starting with release 7.2, all Infor Campaign Management objects have their last modification date recorded in metadata. Metadata can be selected for export by date, with the selection including all metadata objects last modified date is after a selected date.

Note: Object modification dates are only recorded for saved reports in releases prior to 7.2. If you have upgraded from an earlier version, the modification dates of upgraded objects other than reports initially set to the date when you first upgraded to version 7.2 or later. If you import metadata that was exported from a release prior to 7.2, the modification dates of imported objects other than reports will initially be set to the import date.

The Exporting Metadata Dialog Box

The Exporting Metadata dialog box ("Figure 120: Exporting Metadata Dialog Box" on page 431) allows you to export sets of metadata objects to a .mdb file.

Figure 15-1: Exporting Metadata Dialog Box

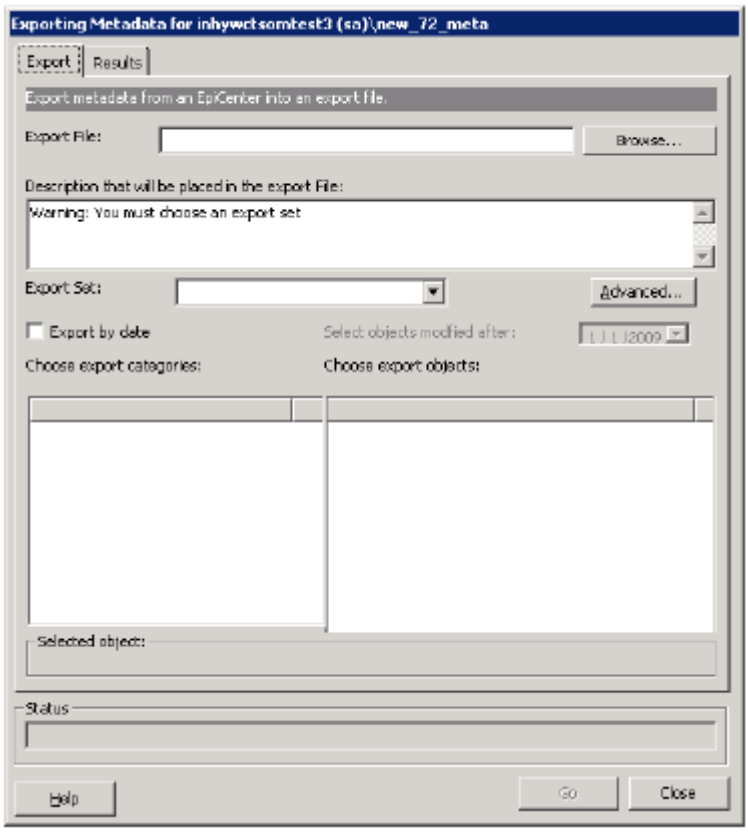


Figure 120: Exporting Metadata Dialog Box

Exporting metadata to a .mdb File

1. In the **Export File** text box, specify the name and location of the export file. You can use **Browse** to navigate to the correct location on your machine.
2. In the **Export Set** drop-down list box, specify the set of metadata objects that you wish to export. When you select a set, export groups that are included in that set are displayed in boldface in the Choose Export Categories pane. When you click the checkbox next to a group, you change the selection state of all of the categories within that group. You can specify individual export categories within a group by selecting the checkbox next to each category that you wish to export.

   For descriptions of the export set options, choose **Tools > Miscellaneous > Describe_export** from the menu bar.
3. Check the **Export by date** option to select objects for export based on their last modification date (or the upgrade or import date for objects that are upgraded or imported from a release prior to 7.2). When this option is selected, you can set a date under **Select objects modified after** , and objects in the export set as modified prior to the chosen date are then deselected, with only objects modified on or after the chosen date selected. If you wish, you can modify the selection (adding objects outside the date range and removing objects inside the date range) after choosing a date.
4. Select individual metadata objects within a category by selecting the category in the Choose Export Categories pane, and then selecting the objects that you wish to export in the Choose objects pane. When you click another category, Admin Manager preserves the selections that you have checked in the Choose Export Objects pane, even if those choices are no longer displayed.

   Categories and objects that are selected with a gray checkbox include only a subset of the available objects under that category.
5. Click **Advanced** and deselect **Export related objects** (which is selected by default) if you do not wish to export objects that are related to the objects that are selected in the Choose Export Categories pane. For example, by default, if you export an attribute layout, the attributes that are included in that layout are also exported. If **Export related objects** is deselected, attribute layout is exported.
6. Click **Advanced** and select **Clear edit flags** if you wish to specify that all exported metadata must be marked as unedited in the export file, regardless of whether it is edited in Admin Manager.

   When you drop edit flags, objects included in the export file are not imported into an EpiMeta database that already contains these objects, unless you deselect the **Preserve edits** option in the advanced import options dialog box. If built-in objects with cleared edit flags are imported into an EpiMeta database, future exports from that EpiMeta database do not have these built-in objects selected for export, since exports normally only include built-in objects if they have been marked as edited.
7. Click **Go**.

   The **Results** tab displays a log of status messages that show the progress of the export operation, which you can save in case an error occurs.

Importing Metadata

When importing metadata into an existing data mart, the import operation detects and attempts to overwrite existing metadata. Existing top-level objects are overwritten in place. The children of the top-level objects are deleted before being recreated. The definition of “existing” is usually based on a

unique name column for one of the metadata tables. For instance, Web pages must have unique names, so an attempt to import a Web page with an existing name results in a warning message (unless the **Always Replace Existing Data** option is selected).

Reimporting metadata objects does not delete references to those objects. For example, a measure definition can be reimported without deletion of the measure layouts that point to that measure.

Import operations are atomic. The entire import operation must be completed before the EpiMeta database can be changed. Partial changes specified in an EpiMeta database as part of an aborted import operation are rolled back.

Note: Import operations can overwrite existing metadata. Infor strongly recommends that you export a backup copy of the metadata for any production system, before you import metadata into that system.

The Importing Metadata Dialog Box

The Importing Metadata dialog box ("Figure 121: Importing Metadata Dialog Box" on page 433) allows you to import selected objects or categories from an export file.

Figure 15-2: Importing Metadata Dialog Box

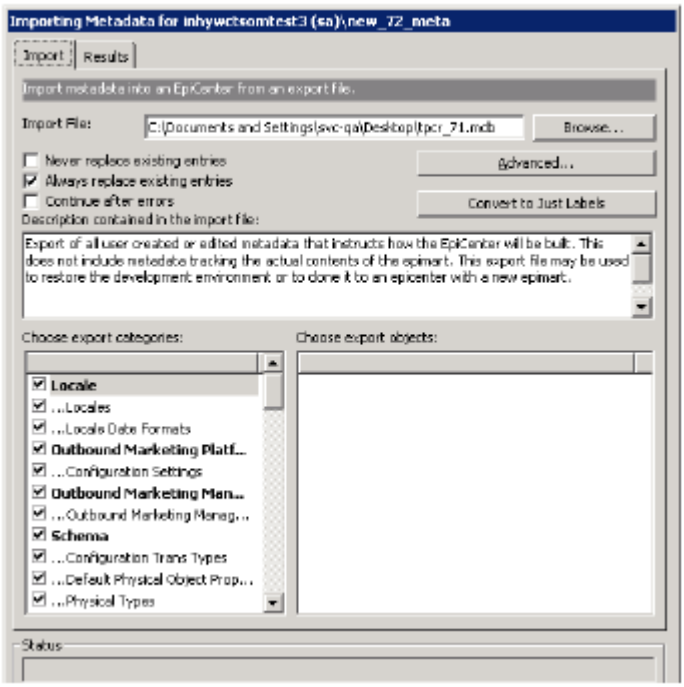


Figure 121: Importing Metadata Dialog Box

Importing metadata to a .mdb file

1. Click the **EpiCenter** menu and select **Import > Import Metadata**.
2. Choose the import .mdb file in the file selection window that opens and click **OK** .

The Importing Metadata dialog box opens with the path to the import file that you chose shown in the **Import File** text box.

- The Description contained in the import file pane displays descriptive information about the import file, including the name of the source EpiMeta, the export user, export time, and the selected export set.
- The Choose Export Categories pane displays the object categories to be imported. All of the object categories in the export file are selected by default, but you can deselect them to leave them out of the import operation.

3 Click an object category in the Choose Export Categories pane to display the objects that fall under that category in the Choose Objects pane. All of the objects in the category are selected by default, but you can deselect them to leave them out of the import operation. Categories and objects that are selected with a gray checkbox include only a subset of the available objects under that category.

Table describes the additional options that you can specify in this dialog box. Some options are available only when you click the **Advanced** button.

4 Click **Go** after you finish selecting your options.

The **Results** tab displays a log of progress messages.

Table 85: Importing Metadata Dialog Box Options

Option	Description
Never/Always Replace Existing Entries	Choose Always if metadata objects from the export file are to replace objects of the same name and type that are already present in the EpiMeta database. Choose Never if you wish to use the existing objects instead. This option affects built-in metadata as well as custom metadata.
Continue After Errors	When checked, this option specifies that the import operation is to continue despite encountering errors. Otherwise, Admin Manager prompts for confirmation before continuing whenever it encounters an error.
Convert to Just Labels	When clicked, causes only the labels in the Import file to be imported. Use this option when importing translation .mdb files.
Advanced > Preserve Edits	If selected, an imported object which is not edited not replaces an existing object of the same name that is edited.
Advanced > Merge schema objects	When checked, an imported fact or dimension with the same name as an existing fact or dimension is merged into the existing table rather than replacing it with a destructive import operation. See Also, "Schema Merging" on page 437.

Option	Description
Advanced > Merge Topics	When checked, an imported topic with the same name as an existing topic is merged into the existing topic rather than replacing it with a destructive import operation. The resulting topic will have the nodes and links of the imported topic plus any nodes and links in the existing topic that are not present in the imported topic.
Advanced > Destructive Import	When checked, a destructive import operation is performed. See Also, "Destructive Import" on page 436.
Advanced > Just Labels	When checked, only object labels are imported. The actual object definitions in the export file are not imported.
Advanced > Update Statistics	If selected, EpiMart statistics that are used for AggBuilder, MomBuilder and other related queries are updated in the database.
Advanced > Logging > Allow unlimited messages	When checked, this option specifies that the import operation must continue regardless of the number of error messages or warnings that are generated. If you do not select this option, use the Messages allowed without aborting parameter to specify how many messages can be generated before the import operation aborts.
Advanced > Logging > Enable Detailed Logging	When checked, the import logs will contain more details of the operations performed during import. This can result in significantly larger log files but can aid in debugging.
Advanced > Logging > Edited Object Skip	When Warn is selected, a warning dialog is displayed about objects that are not imported due to edits that are specified on existing objects in your current EpiCenter. Specify List if you wish to write these warnings to the Import log file.
Advanced > Logging > Built-in Type Skip	When Warn is selected, a warning dialog is displayed about rows that are skipped because of a built-in type change. Specify List if you wish to write these warnings to the Import log file.

Option	Description
Advanced > Logging > Overwrites	When Warn is selected, a warning dialog is displayed about all named objects that are overwritten during the operation. Specify List if you wish to write these warnings to the Import log file.
Advanced > Logging > Parent Not Imported	When Warn is selected, a warning dialog is displayed about related objects that are not imported with an object. For example, if an attribute is not imported, then all filter groups and filter elements that belong to the attribute are also not imported. If you wish to warn that these child objects as not being imported, select this option. Specify List if you wish to write these warnings to the Import log file.
Advanced > Logging > Missing Optional Reference	When Warn is selected, a warning dialog is displayed about optional foreign keys that are dropped during import. For example, if you import an attribute that references a dimension column that does not exist in the new EpiCenter, the reference is dropped and replaced with NULL. If you wish to log this type of occurrence, select this option. Specify List if you wish to write these warnings to the Import log file.
Advanced > Logging > Other Warnings	Specify List if you wish to write any other class of warnings to the Import log file.

Destructive Import

In a destructive import operation, all non-imported objects in some categories (including non-essential built-in objects) are deleted after objects in that category are imported from the export file.

For example, if you are importing attributes into an EpiMeta database with this option selected, the EpiMeta database contains only the essential built-in attributes that are created during EpiCenter initialization and the attributes found in the export file.

Related objects that are owned by the deleted object are also deleted during this operation. For example, when deleting an attribute, the attribute's filter groups and filter elements are also deleted.

The following export categories participate in destructive import:

- Attributes

- Attribute Layouts
- Delegates (entries that record which user or group has the ability to act on behalf of another user or group)
- Fact Terms
- LDAP Servers
- Link Categories
- Measures
- Measure Layouts
- Measure Sets
- Measure Units (from the Configuration dialog box)
- Strings (from the Configuration dialog box)
- Web pages (Ticksheets)
- Topics
- Transaction Filters
- Transaction Type Sets
- Users
- User Groups

Schema Merging

If you import fact or dimension tables with the same names as facts or dimensions that currently exist in your data mart, a schema merge is performed by default. Schema merging combines columns, aggregates and fact indexes from both the old and new fact or dimension into a new object.

For example, if you import a dimension named “Customer” into a data mart that already has a Customer dimension, the resulting Customer dimension contains the UNION of the columns in both dimensions. An imported column that has the same name as a column in the existing fact or dimension overwrites the old column.

Imported fact indexes and dimension and fact aggregates are also merged with existing indexes and aggregates. If there is a name clash between an old and new dimension aggregate or fact aggregate group, the new aggregate or aggregate group overwrites the old.

If you perform a destructive import operation when you are importing schema objects with conflicting names, deselect the **Merge schema objects** option in the **Advanced** area of the Import dialog box. Only imported columns, aggregates and indexes exist after a destructive import operation.

Exporting and Importing Transaction Types

It is not possible to have two transaction types with the same integer value. If you attempt to import a transaction type that uses the same number as a different existing transaction type, the new type is not imported.

Exporting and Importing Jobs

When you export an extraction job, all job steps in the job are exported, but the global extraction steps that are used by these job steps are not automatically exported. If such an exported job is then imported into another EpiCenter, the imported job makes use of the extraction steps in that EpiCenter, if possible. If no matching extraction steps are available in the target EpiCenter, then the job will be in an inconsistent state.

If you wish to export the global extraction steps that are used in an extraction job, then you must select those steps in the Exporting Metadata dialog box when you export the extraction job.

Exporting and Importing Localized Data

If you wish to import a .mdb file from one EpiCenter to another EpiCenter, change the locale of the source EpiCenter to match the locale of the destination EpiCenter before you create your export file.

If you import a .mdb file that was created in one language into an EpiCenter that has labels defined in another language, the imported labels are not visible because they are defined for the wrong locale. In addition, Scrutiny generates warnings about missing labels for the newly imported objects.

For additional information about multi-language EpiCenters, see "Multi-Lingual EpiCenters" on page 78.

Exporting and Importing Topics

If you export and import a topic, either by using the export/import commands, or by dragging and dropping a topic between two EpiMeta databases in Admin Manager, user permissions and saved reports for that topic are not preserved.

Use the following steps to preserve user permissions and saved reports when exporting and importing a topic.

Preserving User Permissions and Saved Reports during Import/Export

1. Rename the topic before exporting (for example, Development-Sales instead of Production-Sales).
2. Export the topic and deselect **Export-related-objects**.
3. Import the topic to the new EpiMeta database.
4. Set the template topic to be the topic that is copied and imported.
5. Click **Topic Update**.

Export File Format

Each Microsoft Access Export database has the same schema. This schema can be thought of as a metaschema for representing relational data. For a description of these tables, see the table below.

To modify the row contents contained in an export file, edit the `Export_col` table, which is simply a collection of name-and-value pairs for columns.

Table 86: Export Tables

Table Name	Description
Export_col	Contains one row per column per row of metadata being exported. Only non-relationship columns are contained in this table.
Export_rel	Contains one row per relationship between two rows of metadata. Can be a relationship between two rows contained in the Export file, or between one row in the Export file and one reference to a row in a foreign EpiMeta.
Export_row	Contains one row per metadata row being exported.
Export_status	Contains header information about the Export file.
Export_tbl	Contains one row per metadata table being exported.
Rel_parent	Contains a reference to a metadata row in the foreign EpiMeta.

Note: Incorrect edits to an exported .mdb file can result in a file that cannot be imported or that corrupts the target metadata when imported.

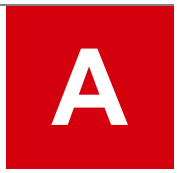
Migrating Materialized Lists

Users can create materialized lists by choosing the **Generate and save list members** option when saving lists in the Report Gallery. When a user saves a materialized list, the membership of that list does not change as the contents of the data mart are subsequently updated. By contrast, when a user saves the selection criteria for a list, the list membership is recreated whenever the list is cited in a report or campaign.

Because materialized lists can be large, Admin Manager does not export them. Instead, it only exports the saved reports that contain the selection criteria for those lists. To migrate materialized lists from one EpiOp database to another, you can use the backup-and-restore or bulk-loading facilities of your database server to copy all of the tables with names of the form `list_*` to another EpiOp database.

To avoid copying list tables that are no longer being used, you can use the Purge facility in Admin Manager (see "Purging EpiMart and EpiOp Tables" on page 375) to remove outdated lists before migrating materialized lists.

Infor Campaign Management Macros



This appendix describes the built-in macros for SQL and operating-system commands supplied with Infor Campaign Management. You can use Infor Campaign Management macros in extraction jobs, queries against your EpiCenter, and the command line for the Infor Campaign Management Server. You can also define your own macros with Admin Manager.

Note: Do not copy and paste scripts, commands or code from this document. Line breaks and some other characters picked up when you copy from the PDF can cause errors in the pasted text.

SQL Macro Syntax

The syntax for an Infor Campaign Management SQL macro takes the following general form:

- Macro references begin with a pair of dollar signs.
- Arguments to SQL macros are enclosed within square brackets. The separator for arguments is a comma surrounded by tilde characters.
- White space between the macro reference and the opening square bracket of the argument list is not allowed.

The following example expands correctly:

```
$$NVL[MAX(col_1)~,~0]
```

The following example does not expand correctly:

```
$$NVL [MAX(col_1)~,~0]
```

The Infor Campaign Management macro interpreter allows white space between arguments and argument separators. However, those white-space characters are passed through and often appear

as part of the expanded SQL statement. Take care to ensure that any white-space characters you embed in your arguments list do not adversely affect the resulting SQL syntax.

"Table 88: Extraction Set Identification Macros" on page 480 illustrates the syntax to use for the SQL macros.

```
$$MACRO
```

```
$$MACRO[argument  ]
```

```
$$MACRO [argument1~,~argument2 ~,~...~,~argumentn]
```

```
$$MACRO_object~.~MACRO_attribute_name
```

Macro Values and Objects

In most contexts, there is an implicit concept of a working object that is used in macro expansion. In extraction code that is used to populate a fact or dimension staging table, the working object corresponds to the fact table or the dimension table that is being populated. For macros that expand to attribute names of working objects, $$ macro_name is equivalent to <working_object>~.~macro_name. When implementing complex extraction logic, you can use macros to find values of Infor Omni-Channel Campaign Management system objects, as described in "Working Objects" on page 487.

For example, as shown in "Table 90: Attributes in a Fact Table Object" on page 488 for a fact table that is currently in context, `$$FCTTBL` expands to the name of the fact table, which is a string. `$$FCTCOL` expands to the list of all fact columns in the fact table, which is a collection of strings. `$$DIMTBL` expands to the list of all dimension roles contained in the fact table, which is a collection of objects .

A macro can expand to a string or to a list, which can contain strings or lists of attributes. A macro can be any value , where a value is defined as follows:

- A value is a string, a collection , or an object .
- A collection is an ordered list of strings or objects .
- An attribute is a string = value pair.

- An object is a list of attributes .

The following syntax is used to reference the attributes:

```
macro_expression~.~attribute_name
```

If macro_expression does not expand to an object value, then a run-time error is generated. attribute_name is the string portion of an attribute of the object. The result of expansion is the value of the attribute.

For data mart state macros (see "Table "Data Mart Macros" on page A-39," on page 485), $$ MACRO i s equivalent to $$ MACRO[table_name~,~table_type] , where table_name is the name of the working object table and table_type is the type of table (usually `Fact` or `Dimension`). Thus, in the context of a fact table called `my_fact_table` , `$$CURR` is equivalent to `$$CURR[my_fact_table~,~Fact]` .

Macro Loops

Most extraction SQL code uses only the basic database-independent macros. In some cases can perform more complex expansions. Infor Campaign Management loop macros expand to sequences of strings. This allows you to do such things as applying a single statement to each column of a table, without knowing the number of columns in advance.

The `$$FOREACH` loop macro has the following syntax

```
$$FOREACH [ loop_variable_name ~,~ collection_expression ~,~ loop_body
 { ~,~ maximum_repetitions } ]
```

Note that:

- collection_expression is expanded before the loop begins and it must expand to a collection value.
- loop_variable is a string which can be used to refer to the current element in the collection.
- loop_body is a string containing the macro expression to be expanded for every element in the collection.
- maximum_repetitions is an optional argument that limits the loop to the first maximum_repetitions elements of the collection.

If collection_member is a member of the collection_expression collection, then the instantiated loop body for collection_member consists of loop_body with all occurrences of $$ loop_variable_name replaced by collection_member . The result of `$$FOREACH` is the concatenation of all instantiated loop bodies.

You can use the $$FOREACH_DELIM macro to insert delimiter strings, such as semicolons, between the instantiated loop bodies:

```
$$FOREACH_DELIM[ delimiter ~,~ loop_variable_name ~,~ collection_
expression ~,~ loop_body { ~,~ maximum_repetitions } ]
```

delimiter is a string that is inserted between the instantiated loop bodies.

For example, the following loop enumerates all the fact columns of a fact table:

```
$$FOREACH[COL ~,~ $$FCTCOL ~,~ $$COL]
```

The following nested loops create all the indexes for a fact table:

```
$$FOREACH_DELIM[IND~,~$$FCTIND ~,~
```

```
CREATE INDEX_SIMPLE_OBJ[ $$FCTTBL[]$$NEXT$$[]_$$IND ~,~$$FCTIND~,~ $$
FCTTBL[]$$NEXT ~,~ . . . ~,~ ~,~Y~,~N~,~FACT_IDX~, ~$$FCTNAME[]~,~$$
NEXTAB]
```

```
]
```

Each $$FOREACH or $$FOREACH_DELIM instance opens a new scope for its loop variable. That is, the definition of the inner loop variable macro supersedes any outer loop variable macros with the same name. The scope of a variable is closed when the expansion of the loop is complete.

For example, given the following nested loops:

```
$$FOREACH[ X ~,~ $$FCTTBL ~,~  $$X $$FOREACH[X ~,~ $$FCTTBL ~,~ $$X
... ] ... $$X]
```

The first and the third $$X occurrences expand to the current value of the outer loop, but the second $$X occurrence expands to the current value of the inner loop.

It is also possible to refer to the iteration number in a loop macro. For any loop variable, if the name of the variable is loopvar , then the loopvar `_TAG` macro expands to the position of the current element in the collection. For example, in a loop macro with a loop variable named `$$MYVAR` , for the fourth element of a collection, `$$MYVAR_TAG` expands to `4` .

If you are defining a new macro, you can use the `$$ALL` and `$$TO_COLLECTION` macros to write a macro that takes a variable number of arguments. For example, you can write a variable-argument `$$COALESCE` using these macros as follows:

```
$$FOREACH[X ~,~ $$TO_COLLECTION[$$ALL] ~,~
```

```
 NVL($$X, ] null $$FOREACH[X ~,~ $$TO_COLLECTION[$$ALL] ~,~ ) ]
```

Given this definition, `$$COALESCE[ c1 ~,~ c2 ]` expands to:

```
NVL(c1, NVL(c2, null ) )
```

Macro Blocks

Some macros are used only in the context of one of the following kinds of code blocks:

- A CASE Block is a block of code that begins with a `$$CASE_BEGIN` macro, possibly followed by one or more `$$CASE_ELSEIF` macro, possibly followed by a `$$CASE_ELSE` macro, and concluding with a `$$CASE_END` macro.
- A DDL Block is a block of code that begins with a `$$DDL_BEGIN` or `$$DDL_BEGIN_NO_DECLARE` macro and ends with a `$$DDL_END` macro. All code that makes changes to the schema must be inside a DDL Block.
- A DECLARE Block is a block of code that consists of a `$$DECLARE_BEGIN` macro followed by a number of `$$DECLARE_BODY` macros. Variables must be declared in DECLARE Blocks. A DECLARE Block must be followed by a `BEGIN ... END` block or by a DDL Block that starts with `$$DDL_BEGIN_NO_DECLARE` .
- An IF Block consists of an `$$IF` macro followed by an `$$END_IF` macro.

- A SEARCHED_CASE Block is a block of code that begins with a `$$SEARCHED_CASE_BEGIN` macro, possibly followed by one or more `$$SEARCHED_CASE_ELSEIF` macro, possibly followed by a `$$SEARCHED_CASE_ELSE` macro, and concluding with a `$$SEARCHED_CASE_END` macro.

Note: If a table is created in a system-managed tablespace, then indexes for that table must be created in the same tablespace.

Defining a Macro

Your EpiCenter includes a collection of SQL macros, listed in "Built-In SQL Macros" on page 447. You can also use Admin Manager to create your own SQL macros.

Creating New SQL Macros

1 In the Extraction folder, right-click **SQL Macros** and select **New SQL Macro** to display the SQL Macro dialog box.

Figure A-1: SQL Macro Dialog Box

2 In the **General** tab of the SQL Macro dialog box, enter the macro name (without the symbol `$$`).

3 In the **Database Type** list box, select a database and then enter the SQL syntax for that database type. Repeat for all other database types, unless you want the macro to translate to nothing in one database. You might not need to translate a macro if the macro applies only to one vendor (the $$NO_FROM_LIST macro on SQL Server), or if it is specific to a vendor (the $$ORACLE macro).

4 Enter a description of the macro in the **Description** area, if desired.

5 You can use the **Test Area** tab to test macro translations without executing any SQL code. Select whether you want to display the translation for SQL Server, Oracle, or DB2, and click **Show Translation** . In the Results dialog box that is displayed, you can copy the results with the **Copy to Clipboard** button and paste them into the **General** tab.

   When viewing the translated SQL, note that any occurrences of the `$$` symbol are highlighted to make them easy to find. These macros did not translate because they do not exist, often because of a typo or other error.

Built-In SQL Macros

Infor provides a number of built-in SQL macros that allow you to write extraction jobs and queries against your data mart in a database-independent fashion. Where necessary, Infor also provides certain database-specific macros.

Deprecated Macros

The following macros are marked DEPRECATED and must be removed from all extraction code before the next release of Infor Campaign Management:

- $$COUNT_ROWS_SELECT
- $$COUNT_ROWS_FROM

(Replace with $$SELECT_COUNT_* macros.)

- $$TO_DATE
- $$TO_DATEFMT

(Replace with $$TO_DATE_FROM_EPIDATE and $$TO_DATE_FROM_EPITIME.)

- $$CREATE_TABLE_AS_SELECT_DDL_EXEC
- $$CREATE_TABLE_AS_SELECT_SIMPLE_OBJ_DDL_EXEC

(Replace with $$CREATE_TABLE_AS_SELECT_* macros.)

Deleted Macros

The following macros are removed from Infor Campaign Management and are not translatable in this version:

- $$ADD_MACRO
- $$CREATE_INDEX_IF_NOT_EXISTS

- $$DOUBLESTRING
- $$EPIINT
- $$EPIKEY
- $$FACTMONEY
- $$FACTQTY
- $$IDENTITY
- $$MONEYSTRING
- $$REMOVE_MACRO

Updated Macros

The following macros have changed from Version 5.0.x to Version 10.1.0 and must be updated in your existing extraction jobs:

- $$COUNTER
- $$INDEX_PROPERTIES
- $$LENGTH
- $$LENGTH_IN_CHARS
- $$TABLE_PROPERTIES
- $ $TABLESPACE_CLAUSE

Database-Independent Macros

Infor supports multiple database servers for EpiCenters. Infor Campaign Management database-independent macros allow you to isolate your data mart from syntax differences that result from vendor-specific extensions to SQL.

Table 87: SQL Macros

Macros	Description
$$ADD_DAYS[date_expression ~,~ number	Returns a date representation of its first argument plus its second argument as a number of days. For example: `$$ADD_DAYS[$$DBNOW ~,~ 1]` returns a date value that is one day after the current date and time.

Macros	Description
$$ADD_MONTHS[date_expression ~,~ number]	Takes two arguments; adds the second as a number of months to the first argument, which must be a date.
$$ALL	Expands to a string of all arguments to the enclosing macro. The arguments are separated by ~,~.
$$ALTER_INDEX[index_name ~,~ alter_options]	Alters index with passed options clause. Can be used to change an index's degree of parallelism after the index is created.
$$ALTER_TABLE[table_name ~,~ alter_options]	Used to build the syntax for the $$CREATE_TABLE_AS_SELECT and $$CREATE_TABLE macros. This macro can be used to change a tables degree of parallelism after the table is created.
$$ANALYZE_SAMP_MIN_ROWS	The number of rows that can be present in a table before the sampling percentage is used in the $$ANALYZE_TABLE macro. The default value for this macro is 100,000.
$$ANALYZE_SAMP_PCT	The sampling percentage rate for the $$ANALYZE_TABLE macro. Set this macro to an integer value between 1 and 100. Note: This macro cannot be set to a non-integer value. Using non-integer values can result in full table analyze on large tables.
$$ANALYZE_TABLE[table_name]	On an Oracle or DB2 database server, performs a size-based analysis of the indicated table and associated indexes.
$$ANALYZE_TABLE_TBL_ONLY[table_name]	On an Oracle or DB2 database server, performs a size-based analysis of the indicated table.

$$ASSERT_INDEX_EXISTS[index_name]	Causes an SQL error if the index named in argument 1 does not exist. For example:
	`$$BEGIN_ASSERT_INDEX`
	`$$ASSERT_INDEX_EXISTS['XPKCustMap_B]`

Macros	Description

	`$$ASSERT_INDEX_EXISTS['XPKAppMap_B']`
	
	`$$END_ASSERT_INDEX`

$$BEGIN_ASSERT_INDEX	(Oracle only) Declares the DECLARE INDEX_NOT_EXISTS exception. See "$$ASSERT_INDEX_EXISTS[index_name]" on page 449.
$$BEGIN_TRANSACTION	Marks beginning of a transaction.
$$BIGDATE	Declares a BIGDATE data type (to record millisecond-precision timestamps). See "Appendix C, "Data Type Values."" on page 503
$$BIGINT	Declares a BIGINT data type (an 8-byte integer) on DB2 and SQL Server. Useful for large tables for which the row count exceeds the capacity of 4 byte integers. Maps to a number type on Oracle. See "Appendix C, "Data Type Values."" on page 503
$$BOOL_TO_YN[testvalue]	Returns the string N if testvalue equals 0. Otherwise, returns Y. For example: `$$BOOL_TO_YN[6]` becomes Y.
$$CASE_BEGIN[expression ~,~ option1 ~,~ value1]	Begins a case statement that compares an expression to a list of options and returns the value for

Macros	Description

	the first matching option. Also provides a single option-value pair. For example:
	`SELECT`
	`$$CASE_BEGIN[col_name ~,~ opt1 ~,~ val1]`
	`$$CASE_ELSEIF[col_name ~,~ opt2 ~,~ val2]`
	`$$CASE_ELSE[else_val]`
	`$$CASE_END`

$$CASE_ELSE[value]	Fall-through value for a case statement that compares an expression to a list of options and returns the value for the first matching option. See $$CASE_BEGIN.
$$CASE_ELSEIF[expression ~,~ option ~,~ value]	Continues a case statement that compares an expression to a list of options and returns the value for the first matching option. The expression argument must be identical to the expression argument in the associated $$CASE_BEGIN statement. See $$CASE_BEGIN.
$$CASE_END	Ends a case statement that compares an expression to a list of options and returns the value for the first matching option. See $$CASE_BEGIN.
$$CASE_NULL	Returns database null value for all platforms. Used to workaround DB2 issues in handling nulls.
$$CAT	Used as an operator to concatenate two strings. For example: `$$TO_CHAR[table1.col1] $$CAT '-' $$CAT $$TO_CHAR[col2]`

Macros	Description
$$CBIN_VAL[testval ~,~ lowerbound~,~upper-bound ~,~ binletter] $$CBIN_END	Can be used to “bin” numeric values into character buckets. Multiple $$CBIN_VAL macros must be followed by a single $$CBIN_END. If testval is in the range lowerbound to upperbound (inclusive), then the expression yields binletter. For example: `$$CBIN_VAL[7~,~1~,~5~,~'A']$$CBIN_VAL[7~,~6~,~10~,~'B']$$$CBIN_END` returns the value B.
$$CHAR	Expands into a type definition for a single character field. See "Appendix C, “Data Type Values,”" on page 503 for information about the Char data type.
$$CIF[expression1 ~,~ expression2 ~,~ arg1~,~arg2]	Conditional macro. Evaluates to arg1 if expression1 is equal to expression2. Evaluates to arg2 otherwise.
$$CLUSTERED_IDX[Y | N]	Returns the database syntax for clustering indexes. Use Y to cluster the index, use N to not cluster the index.
$$COMMENT[comment]	Marks the passed contents with the appropriate DB comment syntax.
$$COMMENT_BLOCK	Translates to nothing. This macro is used for documenting scripts.
$$COUNT_ROWS_FROM $$COUNT_ROWS_SELECT	DEPRECATED. These macros is removed in future releases of Infor Campaign Management. Can be used to count the number of rows in a table. Uses sysindexes on SQL Server for fastest count. For example: `SELECT $$COUNT_ROWS_SELECT the_count $$COUNT_ROWS_FROM[MyTable ]`

Macros	Description
$$COUNTER[initial_value ~,~ datatype]	Returns sequential row numbers for a result set of the specified data type, starting with initial_value. For example: `SELECT $$COUNTER[ 1 ~,~ $$INT ] the_counter.`
$$CR	Expands to the escape code of the carriage return character.
$$CREATE_DOP[object_type] $$CREATE_DOP[object_type~,~ table_name] $$CREATE_DOP[object_type~,~ table_name~,~agg_info]	(Oracle only) Returns the degree of parallelism to be used for the creation of the specified object, table, or aggregate. object_type is the type of object, table_name is the name of the fact, dimension, or external table, and agg_info is the list order number for a fact aggregate or the name for a dimension aggregate.
$$CREATE_INDEX [index_name ~,~ table_name ~,~ column_list ~,~ index_type ~,~ clustered ~,~ tablespace_clause ~,~logging ~,~ options ~,~ alter_index_postproc]	Creates an index where: • index_name is the name of the index to be created. • table_name is the name of the table on which to index. • column_list is the ordered list of columns on which to index. • index_type is a list of the options that are normally placed between CREATE and INDEX keywords in a create index statement. Valid values are UNIQUE, BITMAP, or nothing for non-unique b-tree indexes. • clustered specifies whether the index is clustered. Possible values are Y or N. This value is ignored for Oracle and DB2 databases. • tablespace_clause is the Oracle clause for specifying the name of tablespace in which to create the index. • logging specifies whether logging must be turned on or off. Possible values are Y or N. • options is a list of all object characteristic options, except for tablespace, to be used in index creation. • alter_index_postproc is a list of the alter index options to be used during post processing.

Macros	Description

	For example:
	`$$CREATE_INDEX[`
	`my_dim_stage_idx ~,~ my_dim_stage_table ~,~ column1, column2 ~,~ unique ~,~ N ~,~ $$TABLESPACE_CLAUSE[DIM_STAGE_IDX ~,~ A ] ~,~ N ~,~ $$INDEX_PROPERTIES[DIM_STAGE_IDX ~,~ A] ~,~ $$QUERY_DOP_CLAUSE[ DIM_STAGE_IDX] ]`

$$CREATE_INDEX_NOPP [index_name ~,~ table_name ~,~ column_list ~,~ index_type ~,~ clustered ~,~ tablespace_clause ~,~ logging ~,~ options]	Create an index without post-processing. Arguments are the same as for $$CREATE_INDEX.
	For example:
	`$$CREATE_INDEX_NOPP[ my_dim_stage_idx ~,~ my_dim_stage_table ~,~ column1, column2 ~,~ unique ~,~ Y ~,~ $$TABLESPACE_CLAUSE[DIM_STAGE_IDX ~,~] ~,~ N ~,~ $$INDEX_PROPERTIES[DIM_STAGE_IDX ~,~ A]`
	`]`

$$CREATE_INDEX_SIMPLE_OBJ[index_name ~,~ table_name ~,~ column_list ~,~ index_type ~,~ clustered ~,~ logging ~,~object_type ~,~ object_name ~,~ mirror_letter]	Create an index for physical object properties referenced by object type and table name. object_type is the type of object, object_name is the table name, and mirror_letter indicates the mirrored version (A or B) of the object. If the object is not mirrored, use an empty argument for the mirror_letter parameter.
$$CREATE_TABLE_AS_SELECT[table_name ~,~ tablespace_clause ~,~ logging ~,~ options ~,~ select_statement ~,~ alter_table_postproc]	Generates create-table-as-select (CTAS) syntax, where: • table_name is the name of the table to be created.

Macros	Description

	• tablespace_clause is database syntax for specifying the tablespace or filegroup in which to create table.
	• logging specifies whether logging must be turned on or off. Possible values are Y or N or “”, which leaves the logging decision to the tablespace on which the table is created.
	• options are all object characteristic options, except tablespace, to be used for the CTAS statement.
	• select_statement is the SELECT statement for the CTAS statement. On an SQL Server database server, this statement must include a $$SELECT_INTO_BODY clause.
	• alter_table_postproc are the alter table options to be used for post-processing.
	On a DB2 database server, if the tablespace name is myname_DATA, then indexes are created in the tablespace myname_IDX. Otherwise, indexes are created in the same tablespace as the table.
	For example:
	`$$CREATE_TABLE_AS_SELECT[ my_dim_stage_table ~,~ $$TABLESPACE_CLAUSE[DIM_STAGE_DATA ~,~] ~,~ N ~,~ $$TABLE_PROPERTIES[DIM_STAGE_DATA ~,~ A] ~,~ SELECT column1,column2,column3 ... $$SELECT_INTO_BODY[ my_dim_stage_table]`
	`FROM my_source_table`
	`WHERE ... ~,~ $$QUERY_DOP_CLAUSE[ DIM_STAGE_DATA] ]`

$$CREATE_TABLE_AS_SELECT_NOPP[table_name ~,~ tablespace_clause ~,~ logging ~,~ options ~,~ select_statement]	Generates create-table-as-select syntax without post-processing. Arguments are the same as for $$CREATE_TABLE_AS_SELECT.

Macros	Description
$$CREATE_TABLE_AS_SELECT_SIMPLE_OBJ[table_name ~,~select_statement ~,~ logging ~,~ object_type ~,~ object_name ~,~mirror_letter]	Generates create-table-as-select (CTAS) syntax for physical object properties referenced by object type and table name. object_type is the type of object, object_name is the table name, mirror_letter indicates the mirrored version (A or B) of the object. If the object is not mirrored, use an empty argument for the mirror_letter parameter.
$$CREATE_TABLE_AS_SELECT_SIMPLE_OBJ_DDL_EXEC[table_name ~,~ select_statement ~,~ logging ~,~object_type ~,~ object_name ~,~ mirror_letter]	DEPRECATED. This macro is removed in future releases of Infor Campaign Management. Generates create-table-as-select (CTAS) syntax for physical object properties referenced by object type and table name, with the additional logic from the $$DDL_EXEC macro. object_type is the type of object, object_name is the table name, mirror_letter indicates the mirrored version (A or B) of the object. If the object is not mirrored, use an empty argument for the mirror_letter parameter. This macro can only be used in a DDL block.
$$DAYS_BETWEEN[date1 ~,~ date2]	Returns the number of days between the datetime values date1 and date2.
$$DB_LOGGING[Y | N]	Returns the database syntax for logged or non-logged operations. Use Y to turn on database logging, use N to turn it off.
$$DB2[expression]	(DB2 only) Expands to expression on a DB2 database server. Expands to nothing if the database is not DB2.
$$DBNOW	Returns the date/time from the database. For example: `SELECT Col1 ss_key, $$DBNOW date_modifiedFROM zork`
$$DDL_BEGIN	Starts a block of code that changes the schema. For example: `$$DDL_BEGIN$$NOT_DEBUG[ $$DROP_TABLE_IF_EXISTS[table] ]$$DDL_END` After a DECLARE block, use $$DDL_BEGIN_NO_DECLARE rather than $$DDL_BEGIN.

Macros	Description
$$DDL_BEGIN_NO_DECLARE	Starts a block of code that changes the schema. Use after a DECLARE block. For example: `DECLARE $$VAR[txnFIXED]  $$VAR-CHAR_50$$EOS$$DDL_BEGIN_NO_DECLARE  $$VAR_ASSIGN_BEGIN[txnFIXED] SELECT  $$TO_CHAR[transtype_key]  $$VAR_ASSIGN_INTO[txnFIXED]  FROM Transtype_O  WHERE name = 'FINV_ADJUST'  $$VAR_ASSIGN_END  ...$$DDL_END`
$$DDL_END	Ends a block of SQL that changes the schema. For example: `$$DDL_BEGIN $$DROP_TABLE_IF_EXISTS[  $$FCTTBL[ ]$$NEXT] $$DROP_TABLE_IF_EXISTS[  $$FCTTBL[ ]_INC] $$DDL_END`

$$DDL_EXEC[statement]	All items in the argument list are evaluated at runtime, not when the statement is parsed. This macro can construct SQL based on the values of variables computed in the same SQL block. For example:
	
	`$$DDL_BEGIN$$DDL_EXEC[CREATE INDEX  X_table1 ON table1  (iss, ss_key, date_key)`
	
	`  ]`
	
	`$$DDL_END`
	
	This macro can only be used in a DDL block.

Macros	Description
$$DECIMAL[precision ~,~ scale]	Declares a decimal data type where precision specifies the number of significant digits, and scale specifies the number of decimal places. For example, $$DECIMAL[9~,~5] declares a decimal data type of 9 significant digits with 5 decimal places. See "Appendix C, “Data Type Values,”" on page 503 for more information about this data type.

$$DECLARE_BEGIN	Starts a DECLARE block in which you define the variables used. For example:
	`$$DECLARE_BEGIN`
	`$$DECLARE_BODY[$$VAR[count_INC] $$EPIINT]$$DECLARE_BODY[$$VAR [count_FC]  $$EPIINT]`
	`BEGIN`
	`$$VAR_ASSIGN_BEGIN[cnt_INC]`
	`  SELECT COUNT(1)     $$VAR_ASSIGN_INTO[cnt_INC]  FROM $$FCTTBL[ ]_INC$$VAR_ASSIGN_END`

$$DECLARE_BODY[argument]	Treats its argument as a variable declaration. See "$$DECLARE_BEGIN" on page 458also. Used as part of a DECLARE block.
$$DEFAULT_PARALLEL_DEGREE	Sets the default parallel degree for database operations.
$$DIM_COL_INTMAP_TBL[dim_name ~,~ col_name ~,~mirror_letter]	For col_name an integer-mapped column in the base dimension dim_name, returns the name of the integer map table for col_name in the mirrored table specified by mirror_letter. You can use the

Macros	Description
	$$CURRAB and $$NEXTAB macros to choose an appropriate data mart letter. For example: `$$DIM_COL_INTMAP_TBL[ Campaign ~,~ Campaign_id~,~ $$NEXTAB[Campaign~.~Dimension]]`

$$DIM_COL_INTMAP_VIEW[dim_name ~,~ col_name]	For col_name an integer-mapped column in the base dimension dim_name, returns the name of the view that refers to the integer map table for col_name in the current mirrored table. For example:
	
	`SELECT vw.est_cell_revenue`
	
	`FROM cell_0_V tbl, $$DIM_COL_INTMAP_VIEW[cell~,~ est_cell_revenue] vw`
	
	`WHERE tbl.est_cell_revenue = vw.key_int`

Macros	Description

$$DIRECT_PATH_INSERT_HINT	(Oracle only) Expands to a hint for an Oracle database server to use direct path mode. For example:
	`$$ENABLE_PARALLEL_DML`
	`$$BEGIN_TRANSACTION`
	`INSERT $$DIRECT_PATH_INSERT_HINT`
	`INTO my_table`
	`...`
	`$$END_TRANSACTION`
	`$$DISABLE_PARALLEL_DML`

$$DISABLE_PARALLEL_DML	(Oracle only) Disables parallel insert and update operations for this Oracle session.

$$DROP_INDEX[table_name ~,~ index_name]	Drops the index. For example:
	`$$DDL_BEGIN`
	`$$DROP_INDEX[table1 ~,~ index_name]`
	`$$DDL_END`

Macros	Description
	This macro can only be used in a DDL block.

$$DROP_TABLE[table_name]	Drops the table; does not return an error if the table does not exist. For example:
	
	`$$DROP_TABLE[tbl1]`
	
	Do not place this macro in a DDL block as it is the equivalent of the following:
	
	`$$DDL_BEGIN $$DROP_TABLE_IF_EXISTS[tbl1]$$DDL_END`

$$DROP_TABLE_IF_EXISTS[table_name]	Drops the table; does not return an error if the table does not exist. This macro can only be used in a DDL block. For example: `$$DDL_BEGIN $$DROP_TABLE_IF_EXISTS[tbl1] $$DROP_TABLE_IF_EXISTS[tbl2]$$DDL_END`
$$DROP_VIEW_IF_EXISTS[view_name]	Drops the view; does not return an error if the view does not exist. For example: `$$DDL_BEGIN $$DROP_VIEW_IF_EXISTS[view1] $$DROP_VIEW_IF_EXISTS[view2]$$DDL_END` This macro can only be used in a DDL block.
$$ELSE	The start of the negative clause of an IF statement. See "$$IF[condition]" on page 464also.
$$ENABLE_PARALLEL_DML	(Oracle only) Enables parallel insert and update operations for this Oracle session.
$$END_ASSERT_INDEX	Ends a block of checks that indexes exist. See $$ASSERT_INDEX.
$$END_IF	Ends an IF statement. See "$$IF[condition]" on page 464also.
$$END_TRANSACTION	Marks end of a transaction.

Macros	Description
$$EOS	Ends an SQL statement. For example: `SELECT 'PROCESSED', COUNT(1), 1100FROM table1$$EOS`
$$EXEC_SP[proc ~,~ params]	Execute a stored procedure. The proc argument is the procedure name. The params argument is a comma-separated list of parameters to the stored procedure. Note: On DB2, the params argument must be in the form of #parms [type@,@"parmvalue"](...) with the following restrictions: • A procedure call can have no more than 9 parameters. • Parameters must of type INT or of type VARCHAR. • Parameter values must not contain embedded '@,@' or ']'. For example, $$EXEC_SP[epi_runstats~,~2 [VARCHAR@,@my_table][INT@,@0]] translates to CALL epi_runstats 2 [VARCHAR@,@my_table][INT@,@0] This statement is executed by EpiChannel as a prepared statement that is equivalent to the following at a CLP prompt: CALL epi_runstats('my_table', 0)
$$EXTENT[object_type] $$EXTENT[object_type~,~ table_name] $$EXTENT[object_type~,~ table_name ~,~ agg_info]	(Oracle only) Returns the extent size in KB to be used for creating the specified objects. For partitioned fact tables, this is limited to the table-level settings. Partition overrides do not get returned: • object_type is the type of object • table_name is the name of the fact, dimension, or external table • agg_info is the list order number for a fact aggregate or the name for a dimension aggregate.
$$FLOAT	Database SQL syntax for a float column data type. See "Appendix C, "Data Type Values."" on page 503

Macros	Description
$$FOREACH[loop_variable ~,~ collection_expression ~,~ loop_body ~,~ max_loops] $$FOREACH[loop_variable ~,~ collection_expression ~,~ loop_body]	Expands to a sequence of expressions. For example: `$$FOREACH[  COL ~,~ $$FCTCOL ~,~ $$COL]` See "Macro Loops" on page 443, for more details.

$$FOREACH_DELIM[delimiter ~,~ loop_variable ~,~ collection_expression ~,~ loop_body ~,~ max_loops] $$FOREACH_DELIM[delimiter ~,~ loop_variable ~,~ collection_expression ~,~ loop_body]	Expands to a sequence of expressions. For example:
	`$$FOREACH_DELIM[IND~,~$$FCTIND ~,~`
	`CREATE INDEX_SIMPLE_OBJ[ $$FCTTBL[]$$NEXT$$[]_$$IND~,~$$FCTIND~,~$$FCTTBL[]$$NEXT ~,~`
	`. . .`
	`~,~ ~,~Y~,~N~,~FACT_IDX~,~$$FCTNAME[]~,~$$NEXTAB]`
	`]`
	See "Macro Loops" on page 443, for more details.

$$GET_IDENTITY	Database SQL syntax (the column select list) for getting the current value of unique sequence keys for a given metadata table.
$$GET_IDENTITY_SQL	Database SQL syntax for getting the current value of unique sequence keys for a given metadata table.
$$GREATEST_OF_TWO[val1~,~ val2]	Returns the larger of the two values val1 and val2.

Macros	Description
$$HOURS_BETWEEN[date1~,~ date2]	Returns the number of hours between the datetime values date1 and date2.
$$IF[condition]	Performs a conditional action. For example: `$$DDL_BEGIN$$IF[$$VAR[fc_exists] = 0] $$DDL_EXEC[ $$SELECT_INTO_BEGIN[tmp_tbl] SELECT * $$SELECT_INTO_BODY[tmp_tbl] FROM Old_table WHERE 1=0 ]$$ END_IF$$DDL_END`
$$IJ_FROM[table_name1 ~,~ table_name2]	Performs an inner join on two tables. See "$$JOIN_WHERE[join_condition]" on page 466also.
$$INDEX_PARTITIONS[ObjectType ~,~ factOrDimName ~,~ mirror_letter]	(Oracle only) Returns an index partition clause for the specified fact or dimension table object, including per partition object property and tablespace clause. mirror-letter is a required parameter that specifies a specific mirrored table (A or B). If the table is not mirrored, use a blank space for this parameter.
$$INDEX_PROPERTIES _OVERRIDE[createDOP ~,~ extentSize ~,~ pctFree]	Returns database-specific syntax for setting object characteristics during index creation. This macro can be used to build the syntax for the options parameter of the $$CREATE_INDEX macro or the alter_options parameter for the $$ALTER_INDEX macro: • createDOP is the degree of parallelism to use when creating the index. • extentSize is the size of extents to use for the index. • pctFree is the percentage of free space to reserve for index insertions. Parameters can be left null if they are not applicable to the index or the database platform.
$$INDEX_PROPERTIES[ObjectType ~,~ mirror_letter] $$INDEX_PROPERTIES[ObjectType ~,~ factOrDimName ~,~ mirror_letter] $$INDEX_PROPERTIES[ObjectType ~,~ factOrDimName ~,~ factAggListOrd/DimAggName ~,~ mirror_letter]	Returns a database-specific syntax string that is generated from the object, table, or aggregate characteristics stored in the metadata for index attributes (including partition characteristics for Oracle fact tables). • ObjectType is the type of object

Macros	Description
	• mirror_letter indicates the mirrored version (A or B) of the object • factOrDimName is the name of the fact or dimension table • factAggListOrd/DimAggName is the list order number for a fact aggregate or the name for a dimension aggregate. If the object is not mirrored, use an empty argument for the mirror_letter parameter.
$$INSTR[s1 ~,~ s2]	Returns the position of s1 in s2. For example: `$$INSTR['b' ~,~ 'abc' ]` returns 2.
$$INT	Database SQL syntax for an int column data type. See "Appendix C, "Data Type Values."" on page 503
$$IS_NULL	Database specific SQL function for detecting and converting nulls.
$$JOIN_LEFT_OUTER[left ~,~ right]	Produces a condition for outer joining the first argument to the second. For example: `SELECT ...WHERE $$JOIN_LEFT_ OUTER[ t1.c1 ~,~ t2.c2 ]`
$$JOIN_RIGHT_OUTER	Produces an equals sign appropriate for right outer joins Unlike the $$JOIN_LEFT_OUTER macro, no arguments are needed with this macro. For example: `SELECT...WHERE t1.c1 $$JOIN_ RIGHT_OUTER t2.c2`

Macros	Description

$$JOIN_WHERE[join_condition]	Supplies the WHERE clause for a join. For example:
	
	`SELECT col1, col2`
	
	`FROM Table1 s $$LOJ_FROM[ table2 m ~,~ s.iss = m.iss AND s.col2=m.col2] $$LOJ_FROM[table3 d ~,~ m.col1 = d.col1]`
	
	`WHERE 1=1 $$JOIN_WHERE[m.col1=d.col1(+)] $$JOIN_WHERE[s.iss = m.iss (+) AND s.col2 = m.col2 (+)]`

$$LEAST_OF_TWO[val1 ~,~ val2]	Returns the smaller of the two values val1 and val2.
$$LEFT[string ~,~ nChars]	Returns the first n characters from the left side of a given string.
$$LEFT_BRACKET	Expands to the string “ `[` ” (left square bracket).
$$LENGTH[string]	Returns the length of a string in bytes.
$$LENGTH_IN_CHARS[string]	Returns the length of a string in characters. For example: `$$LENGTH['abc' ]` returns 3.
$$LOJ_FROM[join_condition]	Performs a left outer join. must be used with $$JOIN_WHERE.
$$LONGSTRING	Expands to a VARCHAR data type that is used for decimal values in the campaign manager. This string type eliminates null values, which are not

Macros	Description
	allowed in dimension tables. See "Appendix C, "Data Type Values."" on page 503
$$MAX_FACT_BF_ROW_AGE_IN_DAYS	Sets the number of days before backfeed fact rows can be rolled off (see "The Backfeed Roll-Off Job" on page 324).
$$MAX_SYS_DATE	Returns the highest date supported by the database.
$$MINUTES_BETWEEN[date1~,~ date2]	Returns the number of minutes between the date-time values date1 and date2.
$$MODULO[x ~,~ y]	Returns the remainder when x is divided by y. For example: `$$MODULO[7~,~4]` returns 3. Note: You must explicitly use an INT or BIGINT data type for both of these two arguments.

$$NBIN_VAL[testval ~,~ lowerbound ~,~ upper-bound ~,~ binnumber] $$NBIN_END	Can be used to “bin” numeric values into numeric buckets. Multiple $$NBIN_VAL macros must be followed by a single $$NBIN_END. If testval is in the range lowerbound to upperbound (inclusive), then the expression yields binnumber.
	For example:
	`$$NBIN_VAL[7~,~1~,~5~,~1]`
	`$$NBIN_VAL[7~,~6~,~10~,~ 2]`
	`$$NBIN_END`
	return the value 2.

$$NCHAR	Database SQL syntax for a nchar column data type. This macro translates to a char column data type where nchar data types are not supported. See "Appendix C, "Data Type Values."" on page 503

Macros	Description
$$NO_FROM_LIST	Supplies the “dummy” FROM clause needed by some database vendors. For example: `SELECT 'MODIFIED', $$VAR[modified], 1050 $$NO_FROM_LIST$$EOS`
$$NUM_READERS[]	Supplies the number of readers currently defined in Admin Manager. You define the number in the Number of Readers text box in the SQL Statement tab of the Extraction Command dialog box. This macro is used in conjunction with the $$READER_NUM command as follows: `SELECT a, b, c FROM table WHERE row_number % $$NUM_READERS[] = $READER_NUM[]` (Note that this example assumes that you have a column named row_number.) See $$READER_NUM[].
$$NUMBER[precision]	Declares a number where precision is the precision of the number being declared. For example, if you declare a number with $$NUMBER[5], this number can have 5 significant figures. For more information See "Appendix C, "Data Type Values."" on page 503
$$NVARCHAR[length]	Database SQL syntax for a nvarchar column data type. This macro translates to a varchar2 column data type where nvarchar data types are not supported. See "Appendix C, "Data Type Values."" on page 503
$$NVL[expression ~,~ value]	When the first argument is NULL, replace it with the value in the second argument. For example: `$$TO_CHAR[$$NVL[MAX(col1) ~,~ 1]]`
$$OPTIMIZER_HINT[optimizer_hint_string]	(Oracle only) Returns the Oracle optimizer hint parameter enclosed with the /*+ and */ comments syntax.

Macros	Description

$$ORACLE[expression]	Expands to expression on an Oracle database server. Expands to nothing if the database is not Oracle. For example:
	`SELECT COUNT(1)FROM $$SQLSERVER [sysobjects] $$ORACLE[tabs]`
	`$$DB2[syscat.tables]`

$$PCTFREE[object_type] $$PCTFREE[object_type~,~ table_name] $$PCTFREE[object_type~,~ table_name ~,~ agg_info]	Returns the PCTFREE (for Oracle and DB2) or FILLFACTOR (for SQL Server) value to be used for creating the specified object. For partitioned fact tables, this is limited to the table-level settings. Partition overrides do not get returned: • object_type is the type of object • table_name is the name of the fact or dimension table • agg_info is the list order number for a fact aggregate or the name for a dimension aggregate.
$$PCTUSED[object_type] $$PCTUSED[object_type~,~ table_name] $$PCTUSED[object_type~,~ table_name ~,~agg_info]	(Oracle only) Returns the PCTUSED value to be used for the creation of the specified object. For partitioned fact tables, this is limited to the table-level settings. Partition overrides do not get returned: • object_type is the type of object • table_name is the name of the fact or dimension table • agg_info is the list order number for a fact aggregate or the name for a dimension aggregate.
$$QUERY_DOP[object_type] $$QUERY_DOP[object_type~,~ table_name] $$QUERY_DOP[object_type~,~ table_name ~,~agg_info]	Returns the degree of parallelism to be used for queries on the specified object. object_type is the type of object, table_name is the name of the fact or dimension table, and agg_info is the list order number for a fact aggregate or the name for a dimension aggregate.
$$QUERY_DOP_CLAUSE[object_type] $$QUERY_DOP_CLAUSE[object_type~,~table_name] $$QUERY_DOP_CLAUSE[object_type~,~table_name ~,~agg_info]	Returns database-specific syntax that specifies the degree of parallelism for the specified object. This is anticipated to be used with the $$ALTER_TABLE and $$ALTER_INDEX macros. ob-

Macros	Description
	ject_type is the type of object, table_name is the name of the fact or dimension table, and agg_info is the list order number for a fact aggregate or the name for a dimension aggregate.
$$RAISE_EXCEPTION[exception]	Raises the given exception (as a variable on Oracle, as a string on SQL Server or DB2). For example: `$$RAISE_EXCEPTION[MyException]` raises an exception.
$$READER_NUM[]	Supplies the current reader index. This index starts at zero and goes through n-1, where n is the number of readers defined in Admin Manager. See $$NUM_READERS[].
$$RENAME_OBJECT	Rename tables or other database objects. For example: `$$RENAME_OBJECT[ oldtablename ~ ,~ newtablename]`
$$REPLACE_VIEW[view_name ~,~ view_definition]	Creates or replaces the view view_name using the SELECT statement specified in view_definition.
$$REPLACE_VIEW_BATCH[view_name ~,~ view_definition]	Equivalent to $$REPLACE_VIEW, but uses $$EOB rather than $$BLOCK_BREAK statements. This macro is designed for use in initialization scripts.
$$REVISION_PERCENTAGE	Sets the value at which tables are rebuilt rather than revised (see "Swapping Mirrored Tables" on page 192).
$$RIGHT[string ~,~ nChars]	Returns the first n characters from the right side of a given string.
$$RIGHT_BRACKET	Expands to the string “]” (right square bracket).
$$RPDB_IDX_TBSP	Tablespace for Real-Time database indexes.
$$RPDB_PROFILE_CREATE _PARAMS	(Oracle only) Parameters that are used when writing Data Mart Profile data to an Real-Time database that resides on an Oracle database server. The default value is NOLOGGING.

Macros	Description

$$SEARCHED_CASE_BEGIN[expression ~,~ value]	Begins a general case statement. Returns value if expression is true. For example:
	
	`SELECT`
	
	`$$SEARCHED_CASE_BEGIN[ col_name=test1 ~,~ val1]`
	
	`$$SEARCHED_CASE_ELSEIF[ col_name=test2 ~,~ val2]`
	
	`$$SEARCHED_CASE_ELSE[ else_val]`
	
	`$$SEARCHED_CASE_END`
	
	This macro can only be used with SQL Server, Oracle, and DB2 database servers.

$$SEARCHED_CASE_ELSE[else_val]	Fall-through value for a general case statement. Returns else_val if no other conditions are true. See "$$SEARCHED_CASE_BEGIN[expression ~,~ value]" on page 471also. This macro can only be used with SQL Server, Oracle, and DB2 database servers.
$$SEARCHED_CASE_ELSEIF[expression ~,~ value]	Continues a general case statement. Returns value if expression is true and no previous clauses of the case statement are true. See "$$SEARCHED_CASE_BEGIN[expression ~,~ value]" on page 471also. This macro can only be used with SQL Server, Oracle, and DB2 database servers.
$$SEARCHED_CASE_END	Ends a general case statement. See "$$SEARCHED_CASE_BEGIN[expression ~,~ value]" on page 471also. This macro can only be used with SQL Server, Oracle, and DB2 database servers.

Macros	Description
$$SELECT_COUNT[table_name ~,~ statsUpToDate?]	Returns the appropriate database syntax for accurate row counts in tables, where: • table_name is the database table name. • statsUpToDate? is Y if statistics are up to date and N otherwise. Statistics are up to date if the table was just recreated and reanalyzed. Tables and indexes are reanalyzed every time they are manipulated (for example, during semantic job steps). Oracle and DB2 databases return counts more efficiently when current statistics are available. Note: In Oracle and DB2 it is possible to get an out-of-date row count with this macro if you are performing time/change-based reanalyzes to reduce analyze overhead (in incremental jobs, for example).
$$SELECT_COUNT_PART[table_name ~,~ partition ~,~ statsUpToDate]	(Oracle only) Returns the appropriate database syntax for accurate row counts in a partition, where: • table_name is the database table name. • partition is the partition name. • statsUpToDate is Y if statistics are up to date and N otherwise. Statistics are up to date if the table was just recreated and reanalyzed. Tables and indexes are reanalyzed every time they are manipulated (for example, during semantic job steps). Oracle databases return counts more efficiently when current statistics are available. **Note:** It is possible to get an out-of-date row count with this macro if you are performing time/change-based reanalyzes to reduce analyze overhead (in incremental jobs, for example).
$$SELECT_COUNT_PART_APPROX[table_name ~,~ partition]	(Oracle only) Returns the appropriate database syntax for approximate row counts in a partition, where table_name is the database table name and partition is the partition name. This macro must only be used when approximate row counts are acceptable. **Note:** It is possible to get an out-of-date row count with this macro if you are performing time/change-

Macros	Description
	based reanalyzes to reduce analyze overhead (in incremental jobs, for example).
$$SELECT_FOR_UPDATE[selectWithFromClause ~,~ whereClause ~,~ lockingOptions]	Issues a select using update locks as supported in the database platform. For example, Oracle supports no wait, which allows Oracle to return control to the user if the row has already been locked by another user. This macro must be issued in a transaction. For example, on Oracle you can specify the following call: `$$SELECT_FOR_UPDATE[select foo, bar from my_table ~,~ where needs_modified = 1 ~,~nowait]`
$$SELECT_INTO_BODY[table_name]	Creates a table from a SELECT statement. Expands into a CREATE TABLE AS or a SELECT INTO statement. See "$$CREATE_TABLE_AS_SELECT[table_name ~,~ tablespace_clause ~,~ logging ~,~ options ~,~ select_statement ~,~ alter_table_postproc]" on page 454also.
$$SET_TO_NULL_IF[val1~,~val2]	Returns val1 if val1 is different from val2. Returns NULL otherwise.
$$SMALLDATE	Declares a SMALLDATE. See "Appendix C, "Data Type Values."" on page 503
$$SMALLINT	Declares a double-byte integer. See "Appendix C, "Data Type Values."" on page 503

$$SQLSERVER[expression]	Expands into nothing if the database engine is not SQL Server. For example:
	`SELECT COUNT(1)`
	`FROM $$SQLSERVER[sysobjects] $$ORACLE[tabs]`

Macros	Description
$$SUBSTRING[expression ~,~ start ~,~ length]	Returns the substring of expression that begins at index start and is length characters long. For example: `$$SUBSTRING['momentum' ~,~ 3 ~,~ 2]` returns the string me.

$$TABLE_EXISTS_CONDITION[table_name]	Detects if a table exists. For example:
	
	`SELECT COUNT(1)FROM  $$SQLSERVER [sysobjects]  $$ORACLE[tabs]`
	
	`WHERE  $$TABLE_EXISTS_CONDITION[ table_name]`

$$TABLE_EXISTS_QUERY[tablename]	Detects if a table exists.
$$TABLE_PARTITIONS[object_type ~,~ factOrDimName ~,~ mirror_letter]	(Oracle only) Returns a table partition clause for the specified fact or dimension table object, including per partition object property and tablespace clause. mirror-letter is a required parameter that specifies a specific mirrored table (A or B). If the table is not mirrored, use a blank space for this parameter.
$$TABLE_PROPERTIES[object_type ~,~ mirror_letter] $$TABLE_PROPERTIES[object_type ~,~ factDimExtTablename ~,~ mirror_letter] $$TABLE_PROPERTIES[object_type ~,~ factDimExtTablename ~,~ factAggListOrd/dimAggName ~,~ mirror_letter]	Returns a database-specific syntax string that is generated by the object characteristics stored in the metadata for table characteristics (except for tablespace). This macro also returns partition characteristics that are stored in the metadata (Oracle only). System defaults are returned if characteristics are not overridden for the specified object. mirror-letter is a required parameter that specifies a specific mirrored table (A or B). If the table is not mirrored, use a blank space for this parameter.

Macros	Description
$$TABLE_PROPERTIES_OVERRIDE[createDOP ~,~ extentSize ~,~ pctFree]	Returns database-specific syntax for setting object characteristics during table creation: • createDOP is the degree of parallelism to use when creating the table. • extentSize is the size of extents to use for the table. • pctFree is the percentage of free space to reserve for table insertions. Parameters can be left null if they are not applicable to the table or the database platform.
$$TABLE_WITH_PREFIX[database_name ~,~ table_name]	Qualifies a table name with a database or user name. For example: `SELECT source_system_key issFROM $$TABLE_WITH_PREFIX[ $$ METADBNAME ~,~ source_system]`
$$TABLESPACE [object_type~,~ mirror_letter] $$TABLESPACE[object_type~,~ table_name~,~ mirror_letter] $$TABLESPACE[object_type~,~ table_name~,~ agg_info~,~ mirror_letter]	Returns the tablespace of the indicated object. For partitioned fact tables, this is limited to the table-level settings. Partition overrides do not get returned: • object_type is the type of object • mirror_letter indicates the mirrored version (A or B) of the object • table_name is the name of the fact or dimension table • agg_info is the list order number for a fact aggregate or the name for a dimension aggregate. If the object is not mirrored, use an empty argument for the mirror_letter parameter.
$$TABLESPACE_CLAUSE[object_type~,~ mirror_letter] $$TABLESPACE_CLAUSE[object_type~,~ table_name~,~ mirror_letter] $$TABLESPACE_CLAUSE[object_type~,~ table_name~,~ agg_info~,~mirror_letter]	Returns the database-specific syntax that specifies the tablespace for the indicated object. For partitioned fact tables, this is limited to the table-level settings. Partition overrides do not get returned This macro is anticipated to be used with the $$ALTER_TABLE and $$ALTER_INDEX macros: • object_type is the type of object • mirror_letter indicates the mirrored version (A or B) of the object

Macros	Description
	• table_name is the name of the fact or dimension table • agg_info is the list order number for a fact aggregate or the name for a dimension aggregate. If the object is not mirrored, use an empty argument for the mirror_letter parameter.
$$TINYINT	Declares a single-byte integer. See "Appendix C, "Data Type Values."" on page 503
$$TO_BIGINT	Converts a value to a BIGINT.
$$TO_CHAR[expression]	Converts a value to a character string. For example: `$$TO_CHAR[$$NVL[MAX(col1) ~,~ 1]]` As an alternative, you can use the $$TO_CHAR_UNIVERSAL macro, which accepts both numeric and nonnumeric values. In Oracle, the value must be numeric. In DB2, either float or double types cause errors.
$$TO_CHAR_LENGTH[expression ~,~ length]	Same as TO_CHAR, but allows specification of the data type length of the resulting character value on SQL Server.
$$TO_CHAR_UNIVERSAL[expression]	Converts a value to a character string.
$$TO_COLLECTION[arg1~,~ arg2~,~ ...]	Expands to a collection consisting of all arguments to the macro (arg1, arg2, ...).
$$TO_DATE[expression]	DEPRECATED. This macro is removed in future releases of Infor Campaign Management. Converts a value to a database date with the following format: `MM/DD/YYYY HH24:MM:SS` MM is the month in two-digit notation, DD is the two-digit day, YYYY is the year, HH24 is the hour in 24-hour notation, MM is the two-digit minute, and SS is the two-digit second.

Macros	Description
$$TO_DATE_FROM_EPIDATE[<MM/DD/YYYY>]	Converts an EpiDate object of the form MM/DD/YYYY to a Date object. MM is the month in two-digit notation, DD is the two-digit day, and YYYY is the year.
$$TO_DATEFMT[expr ~,~ format]	DEPRECATED. This macro is removed in future releases of Infor Campaign Management. Converts expression to a date type with the appropriate Oracle format (format is ignored on SQL Server and DB2).

$$TO_EPIDATE[expression]	Converts a date to the string format preferred by EpiChannel. This macro must be used for all columns that have physical type SMALLDATE. For example:
	`SELECT col1 ss_key, $$TO_EPIDATE[date_col] date_modified`
	`FROM zork`
	This macro must only be used with date fields (for example, DATETIME on SQL Server, DATE on Oracle, or TIMESTAMP on DB2) as arguments.

$$TO_EPITIME[expression]	Converts a date and time variable to the string format preferred by EpiChannel. This macro must only be used with date fields as arguments. See $$TO_EPIDATE.
$$TO_FLOAT[number]	Converts a number to float format.
$$TO_HHMMSS[datetime_var]	Converts a datetime variable to a string of the form: HHMMSS where HH represents the hour, MM represents the minute, and SS represents the second.
$$TO_INT[expression]	Converts expression to an integer type. On an Oracle database server, expression must be numeric.
$$TO_NUMBER[expression]	Converts an expression to a number. For example: `$$TO_NUMBER['123']`

Macros	Description
	returns 123.
$$TO_TIME[expression]	Converts its argument to a time representation. For example: `SELECT $$TO_TIME[$$DBNOW]`
$$TO_TIME_FROM_EPITIME[<YYYY-MM-DD HH24:MM:SS>]	Converts an EpiTime object of the form YYYY-MM-DD HH24:MM:SS to a Datetime object. MM is the month in two-digit notation, DD is the two-digit day, YYYY is the year, HH24 is the hour in 24-hour notation, MM is the two-digit minute, and SS is the two-digit second.
$$TO_YYYYMMDD[expression]	Converts a date to a YYYYMMDD string.
$$TRANSLATE_BEGIN $$TRANSLATE_VAL[expression ~,~searchval ~,~ translationval ~,~ nestedcall[. . . ~,~ $$TRANSLATE_ELSE[otherterm] . .]] $$TRANSLATE_END	Searches expression for occurrences of searchval and returns translationval if found. If searchval is not found, nestedcall (which must be $$TRANSLATE_VAL or $$TRANSLATE_ELSE) is executed. Note that the $$TRANSLATE_VAL terms must be nested and the optional $$TRANSLATE_ELSE term must be nested inside the final $$TRANSLATE_VAL. For example: `$$TRANSLATE_BEGIN $$TRANSLATE_VAL['abcdef' ~,~ bce ~,~ 'Value1' ~,~ $$TRANSLATE_VAL['abcdef' ~,~ cde ~,~ 'Value2' ~,~ $$TRANSLATE_ELSE['Other' ]]]$$TRANSLATE_END` returns Value2.
$$TRANSTYPE[name]	Returns the transtype number that corresponds to name.
$$TRIMMED_LENGTH[string]	Returns the trimmed length of a string.
$$TRUNCATE_TABLE[table_name]	Truncates the table table_name.
$$UNKNOWN_DATE	Expands to a date of the form MM/DD/YYYY. 01/01/1900 is used by default. To use a different date, change the value of the $$UNKNOWN_DATE macro in Admin Manager. This date must fall outside of the legal date range, as specified during schema generation.

Macros	Description
	If you change this value, you must perform a force rebuild of your data mart if you wish to propagate the change to existing rows.
$$VAR[variable_name]	References a database variable. For example: `SELECT 'PROCESSED', $$VAR [processed], 1100 $$NO_FROM_ LIST$$EOS`

$$VAR_ASSIGN_BEGIN[variable_name]
$$VAR_ASSIGN_INTO[variable_name]
$$VAR_ASSIGN_END
	Assigns values to a database variable. For example:
	`$$VAR_ASSIGN_BEGIN[max_key]`
	`SELECT $$TO_CHAR[$$NVL[MAX(col1) ~,~1]]$$VAR_ASSIGN_INTO[max_key] FROM table2$$VAR_ASSIGN_END`

$$VARCHAR[length]	Declares a variable-width character data type that holds the maximum number of characters specified by length. See "Appendix C, "Data Type Values."" on page 503
$$DEFAULT_REVISION_PERCENTAGE	Returns the system-wide default for revision percentage (the value at which tables are rebuilt rather than revised). This value is set for individual tables in the Options tab of the Fact and Base Dimension dialog boxes.
$$SELECT_COUNT_APPROX[table_name]	Returns the appropriate database syntax for approximate row counts in tables, where table_name is the database table name. This macro must only be used when approximate row counts are acceptable. **Note:** In Oracle and DB2 it is possible to get an out-of-date row count with this macro if you are performing time/change-based reanalyzes to reduce analyze overhead (in incremental jobs, for example)

Extraction Macros

The extraction macros listed in the following table identify source-system data elements, such as tables and columns, or destination data elements.

Table 88: Extraction Set Identification Macros

Macros	Description
$$COLUMN_CURRENT_VALUE[table_name ~,~ column_name]	This macro is equivalent to $$INT_COLUMN_CURRENT_VALUE.
$$COLUMN_FILTER[table_name ~,~ column_name ~,~ alias_name]	This macro is equivalent to $$INT_COLUMN_FILTER.
$$COLUMN_LAST_VALUE[table_name ~,~ column_name]	This macro is equivalent to $$INT_COLUMN_LAST_VALUE.
$$COLUMN_RANGE_FILTER[table_name ~,~ column_name ~,~alias_name]	This macro is equivalent to $$INT_COLUMN_RANGE_FILTER.
$$DATE_COLUMN_CURRENT_VALUE[tableName ~,~ dateColName]	Expands to the value of the indicated Date column as of the start of the current extraction. Can be used to complete a “two-sided” WHERE clause that also uses the $$DATE_COLUMN_LAST_VALUE macro.
$$DATE_COLUMN_FILTER[tableName ~,~ dateColName ~,~aliasTableName]	Expands to a “one-sided” SQL comparison expression that requires that aliasTableName.dateColName be later than or equal to the value in tableName.dateColName as of the start of the last extraction. This extracts “all transactions that are specified in the database since last time.” Although aliasTableName is optional, Infor suggests that you include it.
$$DATE_COLUMN_LAST_VALUE[tableName ~,~ dateColName]	Expands to the value of the indicated Date column as of the start of the last extraction. This expands to a Date, not to an expression, allowing you to make your own expressions.
$$DATE_COLUMN_RANGE_FILTER[tableName ~,~ dateColName ~,~ aliasTableName]	Expands to a “two-sided” SQL comparison expression that requires aliasTableName.dateColName to be later than or equal to the date in tableName.dateColName as of the start of the last extraction, and less than or equal to this value as of the start of the current extraction. This extracts “all transactions that are specified in the database since last time, not including the transactions that change while the current extraction is running.”
$$DATE_CURRENT_VALUE	Expands to the “current date” value as of the start of the current run. This expands to a value, not to

Macros	Description
	an expression, allowing you to make your own expressions. Can be used to complete a “two-sided” WHERE clause that also uses the $$DATE_LAST_VALUE macro.
$$DATE_FILTER[column_name]	Expands to a “one-sided” SQL comparison expression that requires the column to be greater than or equal to the “current date/time” value as of the start of the last run. This extracts “everything since last time.” No table or alias arguments are available to the macro. If the column needs to be qualified, just add the qualifications in the first argument. For example: `$$DATE_FILTER[oo.date_key]`
$$DATE_LAST_VALUE	Expands to the “current date” value as of the start of the last run. This expands to a value, not to an expression, allowing you to make your own expressions. This macro compares datetime values. The column used for the comparison must be declared as a date and time value. Only the date portion of the value is used; the time portion is discarded.
$$DATE_RANGE_FILTER[column_name]	Expands to a “two-sided” SQL comparison expression that requires the column to be greater than or equal to the “current date/time” value as of the start of the last run, and less than or equal to the current time as of the start of the current run. This extracts “everything since last time, but not including that data that changes while the extractions are running.” This macro compares datetime values. The column used for the comparison must be declared as a date and time value. Only the date portion of the value is used; the time portion is discarded.
$$DATETIME_COLUMN_CURRENT_VALUE[tableName ~,~dateTimeColName]	Expands to the value of the indicated DateTime column as of the start of the current extraction. Can be used to complete a “two-sided” WHERE clause that also uses the $$DATETIME_COLUMN_LAST_VALUE macro.
$$DATETIME_COLUMN_FILTER[tableName ~,~ dateTimeColName ~,~ aliasTableName]	Expands to a “one-sided” SQL comparison expression that requires that aliasTableName.date-

Macros	Description
	TimeColName be later than or equal to the value in tableName.dateTimeColName as of the start of the last extraction. This extracts “all transactions that are specified in the database since last time.” Although aliasTableName is optional, Infor suggests that you include it.
$$DATETIME_COLUMN_LAST_VALUE[tableName ~,~dateTimeColName]	Expands to the value of the indicated DateTime column as of the start of the last extraction. This expands to a DateTime object, not to an expression, allowing you to make your own expressions.
$$DATETIME_COLUMN_RANGE_FILTER[tableName ~,~dateTimeColName ~,~ aliasTableName]	Expands to a “two-sided” SQL comparison expression that requires aliasTableName.dateTimeColName to be later than or equal to the date in tableName.dateTimeColName as of the start of the last extraction, and less than or equal to this value as of the start of the current extraction. This extracts “all transactions are specified in the database since last time, not including the transactions that change while the current extraction is running.”
$$DATETIME_CURRENT_VALUE	Expands to the “current date and time” value as of the start of the current run. This expands to a value, not to an expression, allowing you to make your own expressions. Can be used to complete a two-sided WHERE clause that also uses the $$DATETIME_LAST_VALUE macro.
$$DATETIME_FILTER[column_name]	Expands to a one-sided SQL comparison expression that requires the column to be greater than or equal to the current date and time as of the start of the last run. This extracts everything since last time. No table or alias arguments are available to the macro. If the column needs to be qualified, just add the qualifications in the first argument. For example: $$DATETIME_FILTER[oo.date_key]
$$DATETIME_LAST_VALUE	Expands to the “current date and time” value as of the start of the last run. This expands to a value, not to an expression, allowing you to make your own expressions. This macro compares date/time values. The column used for the comparison must be declared as a date and time value.
$$DATETIME_RANGE_FILTER[column_name]	Expands to a two-sided SQL comparison expression that requires the column to be greater than or equal to the current date and time as of the start of the last run, and less than or equal to the current

Macros	Description
	time as of the start of the current run. This extracts everything since last time, but not including that data that changes while the extractions are running. This macro compares datetime values. The column used for the comparison must be declared as a date and time value.
$$INT_COLUMN_CURRENT_VALUE[tableName ~,~ intColName]	Expands to the value of the indicated integer column as of the start of the current extraction. Can be used to complete a “two-sided” WHERE clause that also uses the $$INT_COLUMN_LAST_VALUE macro.
$$INT_COLUMN_FILTER[tableName ~,~ intColName ~,~aliasTableName]	Expands to a “one-sided” SQL comparison expression that requires that aliasTableName.intColName be greater than or equal to the value in tableName.intColName as of the start of the last extraction. This extracts “everything since last time.” Although aliasTableName is optional, Infor suggests that you include it.
$$INT_COLUMN_LAST_VALUE[tableName ~,~ intColName]	Expands to the value of the indicated integer column as of the start of the last extraction. This expands to a value, not to an expression, allowing you to make your own expressions.
$$INT_COLUMN_RANGE_FILTER[tableName ~,~ intColName ~,~aliasTableName]	Expands to a “two-sided” SQL comparison expression that requires aliasTableName.intColName to be greater than or equal to the value in tableName.intColName as of the start of the last extraction, and less than or equal to this value as of the start of the current extraction. This extracts “everything since last time, but not including that data that changes while the extractions are running.”
$$STRING_COLUMN_CURRENT_VALUE[tableName ~,~ strColName]	Expands to the value of the indicated string column as of the start of the current extraction. Can be used to complete a “two-sided” WHERE clause that also uses the $$STRING_COLUMN_LAST_VALUE macro.
$$STRING_COLUMN_FILTER[tableName ~,~ strColName ~,~aliasTableName]	Expands to a “one-sided” SQL comparison expression that requires that aliasTableName.strColName be earlier in the alphabet than or equal to the string in tableName.strColName as of the start of the last extraction. This extracts “everything since last time.” Although aliasTableName is optional, Infor suggests that you include it.

Macros	Description
$$STRING_COLUMN_LAST_VALUE[tableName ~,~ strColName]	Expands to the value of the indicated string column as of the start of the last extraction. This expands to a value, not to an expression, allowing you to make your own expressions.
$$STRING_COLUMN_RANGE_FILTER[tableName ~,~ strColName ~,~ aliasTableName]	Expands to a “two-sided” SQL comparison expression that requires aliasTableName.strColName be earlier in the alphabet than or equal to the string in tableName.strColName as of the start of the last extraction, and later in the alphabet than or equal to this string as of the start of the current extraction. This extracts “everything since last time, but not including that data that changes while the extractions are running.”
$$TIMESTAMP_CURRENT _VALUE	On SQL Server, expands to the timestamp as of the start of the last run. This expands to a value, not to an expression, allowing you to make your own expressions.
$$TIMESTAMP_FILTER[column_name]	On SQL Server, expands to a “one-sided” SQL comparison expression that requires the column to be greater than or equal to the current timestamp as of the start of the last run. This extracts “everything since last time.” This compares timestamps, which actually have no time or date information in them. The column used for the comparison must be declared as a timestamp type, not as a time or date.
$$TIMESTAMP_LAST_VALUE	On SQL Server, expands to the “current timestamp” as of the start of the last run. This expands to a value, not to an expression, allowing you to make your own expressions.
$$TIMESTAMP_RANGE_FILTER[column_name]	On SQL Server, expands to a “two-sided” SQL comparison expression that requires the column to be greater than or equal to the current timestamp as of the start of the last run, and less than or equal to the current time as of the start of the current run. This extracts “everything since last time, but not including that data that changes while the extractions are running.” This compares timestamps, which actually have no time or date information in them. The column used for the comparison must be declared as a timestamp type, not as a time or date.
$$YYYYMMDD_CURRENT _VALUE	Expands to the current date in YYYYMMDD format as of the start of the current run. This expands to

Macros	Description
	a value, not to an expression, allowing you to make your own expressions.
$$YYYYMMDD_FILTER[column_name]	This is the same as a DATE_FILTER except that the column used for comparison is in YYYYMMDD format rather than datetime format. This extracts “everything since the last day, including the last day.”
$$YYYYMMDD_LAST_VALUE	Expands to the current date in YYYYMMDD format as of the start of the last run. This expands to a value, not to an expression, allowing you to make your own expressions.
$$YYYYMMDD_RANGE_FILTER[column_name]	This is the same as a DATE_RANGE_FILTER except that the column used for comparison is in YYYYMMDD format rather than datetime format. This extracts “everything since the last day, including the last day, but not including today.”

Data Mart Management Macros

The data mart management macros listed in the following table specify data elements within your data mart or conditions that might be true with respect to your data mart.

You must initialize your EpiOp and register your EpiMart before using any macros that require a registered table or dimension.

Table 89: Data Mart Macros

Macros	Description
$$CURR	Expands to the _A or _B suffix (with underscores) of the currently active table. If no table is currently active, $$CURR expands to _V. Requires generated schema.
$$CURR[tableName ~,~ tableType]	Expands to the _A or _B suffix (with underscores) of the currently active table. Allows you to reference the active or new EpiMart tables. If no table is currently active, $$CURR expands to _V. Requires initialized EpiMeta and EpiOp.
$$CURRAB[tableName ~,~ tableType]	Expands to the A or B suffix (without underscores) of the currently active table. Allows you to reference the active or new EpiMart tables. Requires initialized EpiMeta and EpiOp.

Macros	Description
$$CURRBF	Expands to the _A or _B suffix of the currently active backfeed tables. Requires the tables to be registered.
$$CURRBF[tableName ~,~ tableType]	Expands to the _A or _B suffix of the currently active backfeed tables. Allows you to reference the active or new EpiMart tables. Requires initialized EpiMeta and EpiOp.
$$CURREXP	Expands to the _P or _Q suffix of the currently active backfeed tables. Requires generated schema.
$$CURREXP[tableName ~,~ tableType]	Expands to the _P or _Q suffix of the currently active backfeed tables. Allows you to reference the active or new EpiMart tables. Requires initialized EpiMeta and EpiOp.
$$DEFAULT	A default value for a SQL Server or Oracle data store. Depending on the context in which the macro is used, it expands to one of the following: • The name of the database-server host to which a connection is established for the extraction job • The name of the database-server instance for the current extraction job • The username by which the connection is established • The password for the user establishing the connection
$$DIMBFT	The name of the backfeed table, if any, for the registered dimension.
$$DIMKEY	The name of the primary key column of the base dimension table for the registered dimension.
$$DIMNAME	The base name of the registered dimension.
$$DIMTBL	The name of the base table (without suffix letters) for the registered dimension.
$$DSTGKEY	The name of the sskey column in the staging table of the registered dimension.
$$DSTGTBL	The name of the staging table for the registered dimension.
$$FCTBFT	The name of the backfeed table, if any, of the registered fact table.

Macros	Description
$$FCTTBL	The name of the physical base table (without letter suffix) of the registered fact table.
$$FSTGTBL	The name of the staging table of the registered fact table.
$$INITIAL_LOAD[arg]	Expands to arg in a job step that has the Initial Load option selected. Expands to the null string otherwise.
$$MAPTBL	The name of the mapping table for the registered dimension.
$$MARTDBNAME	Returns the name of the data mart (EpiMart) database (for SQL Server and DB2) or schema (for Oracle).
$$METADBNAME	Returns the name of the current metadata (EpiMeta) database (SQL Server) or schema (Oracle).
$$NEXT	Expands to the _A or _B suffix (with underscores) of the next fact or dimension table. Requires generated schema.
$$NEXT[tableName ~,~ tableType]	Expands to the _A or _B suffix (with underscores) of the next fact or dimension table. Requires initialized EpiMeta and EpiOp.
$$NEXTAB[tableName ~,~ tableType]	Expands to the A or B suffix (without underscores) of the next table.Requires initialized EpiMeta and EpiOp.
$$NEXTBF	Expands to the _A or _B suffix of the next backfeed tables. Requires generated schema.
$$NEXTEXP	Expands to the _P or _Q suffix of the next backfeed tables. Requires generated schema.
$$NOT_INITIAL_LOAD[arg]	Expands to arg in a job step that does not have the Initial Load option selected. Expands to the null string otherwise.

Working Objects

Depending on the context in which a macro is used, there can be an implicit working object. These objects are discussed in "Macro Values and Objects" on page 442.

Fact Table Attributes

The working object for a fact table contains the attributes shown in the following table:

Table 90: Attributes in a Fact Table Object

DEGKEY	<collection> of <string>	List of degenerate dimensions
DIMTBL	<collection> of <object>	List of dimension roles
FCTBFT	<string>	Backfeed name
FCTBFTINS	<string>	Backfeed insert table name
FCTCOL	<collection> of <string>	List of fact columns
FCTDATE	<string>	Date key (when applicable)
FCTIND	<collection> of <object>	List of fact indices
FCTINS	<string>	Insert table name
FCTNAME	<string>	Logical fact name
FCTPARTITIONED	<string>	1 if the fact is partitioned, 0 otherwise
FDIMTBL	<collection> of <object>	List of versioned, list-producing, fusion-enabled dimension roles
FPSTGTBL	<string>	Fact pre-staging table name
FCTTBL	<string>	Physical name
FSTGTBL	<string>	Staging table name
NFVDIMTBL	<collection> of <object>	List of versioned, list-producing dimension roles that are not fusion-enabled
NO_PROCESS_KEY	<string>	0 if the table has a process_key column (that is, the table contains both states and transactions), 1 otherwise
VDIMTBL	<collection> of <object>	List of versioned, list-producing dimension roles

Derived facts for list-producing dimensions also include the following attributes:

Table 91: Additional Attributes in a List-Producing Dimension Fact Table Object

Attribute name	Type	Meaning
LPDIMNAME	<string>	Dimension base name for the related list-producing dimension

Attribute name	Type	Meaning
LPDIMTBL	<string>	Dimension table name for the related list-producing dimension
LPDIMSSKEYR	<string>	sskey column name for the related list-producing dimension
LPDIMKEYR	<string>	Dimension-role-key-column name for the related list-producing dimension
LPDIMKEY	<string>	Dimension-key-column name for the related list-producing dimension
LPDIMROOT	<string>	The generated derived fact base name without the fact-type suffix (_msg, _com, _ifr, _sed)

Dimension Role Attributes

The object for a dimension role contains the attributes shown in the following table:

Table 92: Attributes in a Dimension Role Object

Attribute name	Type	Meaning
DIMBASE	<string>	Logical base dimension name
DIMKEY	<string>	Dimension key
DIMKEYR	<string>	Dimension key role
DIMNAME	<string>	Dimension base name
DIMRNAME	<string>	Dimension role name
DIMTBL	<string>	Physical name
DPSTGTBL	<string>	Dimension pre-staging table name
DSTGKEY	<string>	Dimension SS key
DSTGKEYR	<string>	Dimension SS key role
DSTGTBL	<string>	Staging table name
FMAPTBL	<string>	Fusion map table name
MAPEXPR	<string>	Key map expression
MAPEXPR_REAL	<string>	Key map expression for REAL key

Attribute name	Type	Meaning
MAPTBL	<string>	Map name
MEMORY_LOOKUP_FLAG	<string>	1 if keys can be mapped in EpiChannel memory, 0 otherwise
UNMAPPED_DIM_FLAG	<string>	1 if the base dimension is un-mapped, 0 otherwise

Fact Index Attributes

The object for a fact index contains the attributes shown in the following table:

Table 93: Attributes in a Fact Index Object

Attribute name	Type	Meaning
FCTIND	<string>	Index identifier
FINDCOL	<collection> of <string>	List of indexed columns

Dimension Table Attributes

The working object for a dimension table contains the attributes shown in the following table:

Table 94: Attributes in a Dimension Table Object

Attribute name	Type	Meaning
DIMBFT	<string>	Backfeed name
DIMCLI	<collection> of <string>	List of indexed dimension columns
DIMCOL	<collection> of <collection>	List of dimension columns. Collection elements expand to column names if no further attributes are specified.
DIMFOLD	<string>	Fusion old table name
DIMINS	<string>	Insert table name
DIMKEY	<string>	Dimension key
DIMNAME	<string>	Logical base dimension name
DIMOLD	<string>	Old table name
DIMPARTITIONED	<string>	1 if the dimension is partitioned, 0 otherwise

Attribute name	Type	Meaning
DIMTBL	<string>	Physical name
DIMTHIN	<string>	Thin table name
DIMUPD	<string>	Update table name
DPSTGTBL	<string>	Pre-staging table name
DSTGKEY	<string>	Dimension SS key
DSTGTBL	<string>	Staging table name
FSTGTBL	<string>	Fusion staging table name
FMAPTBL	<string>	Fusion map table name
KEYDBPHYSTYPE	<string>	Macro string for dimension key physical type
MAPTBL	<string>	Map name
NON_VERSIONED_FLAG	<string>	1 if the dimension is non-versioned, 0 otherwise
UNMAPPED_DIM_FLAG	<string>	1 if the dimension is unmapped, 0 otherwise

Dimension Column Attributes

The object for a dimension column contains the attributes shown in the following table:

Table 95: Attributes in a Dimension Column Object

Attribute name	Type	Meaning
default attribute	<string>	Column name
UNKNOWN	<string>	Default column value
SECONDARY	<string>	1 if the column is a secondary column, 0 otherwise

Database-Specific SQL Macros

This section lists macros and discusses concerns that apply to specific database servers.

Oracle Macros

The macros in the following table apply to Oracle database servers.

Table 96: Oracle-Specific SQL Macros

Macro
$$BEGIN_ASSERT_INDEX
$$CREATE_DOP
$$DIRECT_PATH_INSERT_HINT
$$DISABLE_PARALLEL_DML
$$ENABLE_PARALLEL_DML
$$EXTENT
$$INDEX_PARTITIONS
$$OPTIMIZER_HINT
$$PCTUSED
$$RPDB_PROFILE_CREATE_PARAMS
$$SELECT_COUNT_PART
$$SELECT_COUNT_PART_APPROX
$$TABLE_PARTITIONS

SQL Server Macros

The macros in the table below apply to SQL Server.

Table 97: SQL Server Macros

Macro
$$TIMESTAMP_FILTER[column_name]
$$TIMESTAMP_CURRENT_VALUE
$$TIMESTAMP_LAST_VALUE
$$TIMESTAMP_RANGE_FILTER[column_name]

System-Call Macros

Infor Campaign Management system-call macros allow you to encapsulate operating-system commands in extraction jobs. You typically use system-call macros to pass filenames or user names that are stored in metadata to operating-system commands. Many commands, such as `RENAME` and `DIR`, are unable to read a database, and few commands can read Infor Campaign Management-created structures.

The following table illustrates the syntax to use for the System-Call macros.

```
$$MACRO
```

```
$$MACRO   [argument  ]
```

```
$$MACRO [arg1, arg2 ]
```

You can use data-store roles to refer to specific data stores that are used in the job. Unless stated otherwise, the arguments for a system-call macro are the names of data-store roles that are defined for an extraction job. When a system call is executed, data-store roles allow a macro to be expanded based on the data store that is assigned to the specified data-store role in the job.

If a macro reference does not match the name of a macro that is defined, the extraction job halts with an error. If the argument to a macro is a role name, and the job does not define that role, the extraction job halts with an error.

"Table 98: Infor Campaign Management System-Call Macros" on page 493 lists the Infor Campaign Management system-call macros.

Note: As noted in the following table, some system call macros can only be used when an Infor Campaign Management instance is specified. You can specify an instance by using the --InstanceName option to EpiChannel (see "The EpiChannel Command Line" on page 351) or by selecting an instance in the Execute Job dialog box.

Table 98: Infor Campaign Management System-Call Macros

Macros	Purpose	Description
$$APPSERVERHOST	Registry	Returns the data in the AppServerHost registry key value. This macro can only be used when an instance is specified.
$$APPSERVERPORT	Registry	Returns the data in the AppServerPort registry key value. This macro can only be used when an instance is specified.

Macros	Purpose	Description
$$CHARTSOUTPUTDIR	Registry	Returns the data in the ChartsOutputDir registry key value. This macro can only be used when an instance is specified.
$$DATABASE[role]	Database Login	Translates to the name of the database or instance when the data-store role is associated with a specific database server (that is, not a data store of type ODBC or File). For example: `isql /S $$SERVER[My Role] /U $$USER[My Role] /P $$PASSWORD [MyRole] /d $$ DATABASE[MyRole] /w 300 /i /Q "gen_ tests_run"`
$$DBVENDOR[role]	Database Login	Translates to the vendor of the database server associated with a data-store role.
$$DIRNAME[role]	File ID	Translates to the directory name of the data store associated with role, without the last filename component and without a trailing slash.
$$DSN[role]	Database Login	Translates to the ODBC connection string for the database associated with role. This string is generated even for databases accessed using native APIs.
$$ENV[var]	System	Translates to the value of the Windows or UNIX environment variable var.
$$EPIBIN	Child Procs	The name of the bin directory under the directory specified by the InstanceRootDir Registry key value. This macro can only be used when an instance is specified.

Macros	Purpose	Description
$$EPIREG[val]	Registry	Translates to the data in the registry key value val for the current Infor Campaign Management instance. Components of val must be delimited by dots (that is, by the character “.”). For example, for an instance called my_inst, $$EPIREG[my_key.my_val] translates to the data in the following Windows registry key value: `HKEY_CURRENT_USER\SOFTWARE\Infor\Instances\my_inst\my_key\my_val` This macro can only be used when an instance is specified.
$$FILENAME[role]	File ID	Translates to the filename of the data store associated with role.
$$IF_UNIX command1 [$$ELSE command2] $$ENDIF	OS-Specific Commands	Expands to command1 if the system call is being executed under the UNIX operating system. If the optional $$ELSE clause is included, expands to command2 if the system call is not being executed under the UNIX operating system.
$$IF_WINDOWS command1 [$$ELSE command2] $$ENDIF	OS-Specific Commands	Expands to command1 if the system call is being executed under the Windows operating system. If the optional $$ELSE clause is included, expands to command2 if the system call is not being executed under the Windows operating system.
$$INSTALLROOTDIR	Child Proc	If InstallRootDir is passed as an argument to EpiChannel (see "The EpiChannel Command Line" on page 351), this macro translates as that string.

Macros	Purpose	Description
		If Instance (-i) is passed as an argument to EpiChannel, this macro returns the same value as $$INSTANCEROOTDIR. If neither argument is passed to EpiChannel, an exception is thrown.
$$INSTANCE_NAME	Registry	The name of the instance's Registry subtree. This macro can only be used when an instance is specified.
$$INSTANCEROOTDIR	Child Procs	Returns the data in the InstanceRootDir registry key value. This macro can only be used when an instance is specified.
$$JOB_NAME	Child Procs	The name of the current job.
$$PASSWORD[role]	Database Login	The password for the user of the database-server or host associated with the indicated data-store role. This macro expands to the empty string if you are using Windows integrated security with the specified database server. Note that the use of this macro results in logging of the specified password, which present security issues.
$$PATH[role]	File ID	Translates to the full pathname of the file that is associated with role.
$$PROGRAM_NAME	Child Procs	The name of the current epichannel program.
$$REGISTRY_EPIPATH	Registry	Name of the Infor Campaign Management instance registry key. For example, `HKEY_CURRENT_USER\$$REGISTRY_EPIPATHinstance_name`

Macros	Purpose	Description
		translates to the registry key for the instance instance_name.
$$SSL_FLAG	Expands to -ssl if the value of the Behavior/EM/UseSSL configuration setting is 1. Otherwise, it expands to nothing.	
$$SERVER[role] $$SQLNET[role]	Database Login	Translates to the name of the database server for the data store that is associated with role. For SQL Server, this is the hostname of the computer on which the database server resides. For Oracle, this is the SQLNet ID (SID). SERVER and SQLNET are identical in behavior and can be interchanged. The data-store type must be a specific database server (not ODBC or File.)
$$USER[role]	Database Login	The user name for the database-server or host associated with the indicated data-store role. This macro expands to the empty string if you are using Windows integrated security with the specified database server.

Date Dimension Fields

B

The table below describes the default date dimension fields that the EpiCenter uses. Columns marked as Required cannot be removed from the Date dimension.

Table 99: Date Dimension Fields

Dimension Column Name	Physical Type	Description
cq_and_cy_name	VARCHAR_50	Calendar quarter and year name. For example, Q1 1999.
cq_name	VARCHAR_50	Calendar quarter name. For example, Q1.
cy_name	VARCHAR_50	Calendar year name. For example, 1999.
day_cq_begin	TINYINT	Whether or not this is a day on which a calendar quarter begins. (0 or 1).
day_cq_end	TINYINT	Whether or not this is a day on which a calendar quarter ends. (0 or 1)
day_cy_begin	TINYINT	Whether or not this is a day on which a calendar year begins. (0 or 1)
day_cy_end	TINYINT	Whether or not this is a day on which a calendar year ends. (0 or 1)
day_fq_begin	TINYINT	Whether or not this is a day on which a fiscal quarter begins. (0 or 1)
day_fq_end	TINYINT	Whether or not this is a day on which a fiscal quarter ends. (0 or 1)

Dimension Column Name	Physical Type	Description
day_fy_begin	TINYINT	Whether or not this is a day on which a fiscal year begins. (0 or 1)
day_fy_end	TINYINT	Whether or not this is a day on which a fiscal year ends. (0 or 1)
day_month_begin	TINYINT	Whether or not this is a day on which a month or period begins. (0 or 1)
day_month_end	TINYINT	Whether or not this is a day on which a month or period ends. (0 or 1)
day_name	SMALLDATE	The day as a native date type.
day_number_in_cq	TINYINT	The number of this day in the calendar quarter, starting at 1.
day_number_in_cy	SMALLINT	The number of this day in the calendar year, starting at 1.
day_number_in_fq	TINYINT	The number of this day in the fiscal quarter, starting at 1.
day_number_in_fy	SMALLINT	The number of this day in the fiscal year, starting at 1.
day_number_in_month	TINYINT	The number of this day in the month or period, starting at 1.
day_number_in_week	TINYINT	The number of this day in the week, starting at 1.
day_number_til_end_cq	TINYINT	The number of days until the end of the calendar quarter, ending with 1. See page "Generating the Schema" on page 161.
day_number_til_end_fq	TINYINT	The number of days until the end of the fiscal quarter, ending with 1. See page "Generating the Schema" on page 161.
fq_and_fy_name	VARCHAR_50	Fiscal quarter and year name. For example, Q1 1999.
fq_name	VARCHAR_50	Fiscal quarter name. For example, Q1.
fy_name	VARCHAR_50	Fiscal year name. For example, 1999.

Dimension Column Name	Physical Type	Description
julian_counter	INT	A counter that assigns sequential integer values to the days in the Date dimension. The first day in the Date dimension has a value of 1.
month_and_cy_name	VARCHAR_50	The name of the month abbreviated and the calendar year. For example, Apr 1999.
month_and_fy_name	VARCHAR_50	Month (abbreviated) or period name and fiscal year. For example, Apr 1999.
month_name	VARCHAR_50	The three-letter abbreviations of the month (without a year or any other value). For example, Apr.
month_number	SMALLINT	Month or period number since the first date in the system, starting at 1.
month_number_in_cq	TINYINT	Month number since the start of the calendar quarter, starting at 1.
month_number_in_cy	TINYINT	Month number since the start of the calendar year, starting at 1.
month_number_in_fq	TINYINT	Month or period number since the start of the fiscal quarter, starting at 1.
month_number_in_fy	TINYINT	Month or period number since the start of the fiscal year, starting at 1.
month_number_til_end_cy	TINYINT	Number of months or periods until the end of the calendar year, ending with 1. See page "Generating the Schema" on page 161.
month_number_til_end_fy	TINYINT	Number of months or periods until the end of the fiscal year, ending with 1. See page "Generating the Schema" on page 161.
vacation_day	TINYINT	Whether or not this day is a vacation day. (0 or 1)
week_friday	SMALLDATE	Date (as a string) for the Friday of the current week. For example, Apr 01 1994.

Dimension Column Name	Physical Type	Description
week_monday	SMALLDATE	Date (as a string) for the Monday of the current week. For example, Apr 01 1996.
week_number	SMALLINT	Week number since the first date in the system, starting at 1.
week_number_cq	TINYINT	Week number since the start of the calendar quarter, starting at 1.
week_number_cy	TINYINT	Week number since the start of the calendar year, starting at 1.
week_number_fq	TINYINT	Week number since the start of the fiscal quarter, starting at 1.
week_number_fy	TINYINT	Week number since the start of the fiscal year, starting at 1.
week_number_til_end_cq	TINYINT	Week number until the end of the calendar quarter, ending with 1. See page "Generating the Schema" on page 161.
week_number_til_end_cy	TINYINT	Number of weeks until the end of the calendar year, ending with 1. See page "Generating the Schema" on page 161.
week_number_til_end_fq	TINYINT	Week number until the end of the fiscal quarter, ending with 1. See page "Generating the Schema" on page 161.
week_number_til_end_fy	TINYINT	Number of weeks until the end of the fiscal year, ending with 1. See page "Generating the Schema" on page 161.
week_saturday	SMALLDATE	Date (as a string) for the Saturday of the current week. For example, Apr 01 1995.
week_sunday	SMALLDATE	Date (as a string) for the Sunday of the current week. For example, Apr 02 1995.
weekday	VARCHAR_50	Weekday prefix; for example, Sun.

Data Type Values

C

This appendix defines the database-type translations for the data types that you select in the Base Dimension, Fact, and External Table dialog boxes in Admin Manager. The data type that you select is replaced by its associated database type. The translation of data type to database type depends on your RDBMS platform.

You can define new data types in the **Data Types** tab of the Configuration dialog box. See "The Data Types Tab" on page 157 for more information.

Table 100: Database Translations for Data Type Values

Data Type	SQL Server	Oracle	DB2
BIGDATE	DATETIME	DATE	TIMESTAMP
BIGINT	BIGINT	NUMBER(19)	BIGINT
CHAR_n	CHAR(n)	CHAR(n)	CHAR(n)
DECIMAL_x_y	DECIMAL(x,y)	NUMBER(x,y)	DECIMAL(x,y)
FLOAT	FLOAT(24)	NUMBER	FLOAT(24)
INT	INT	NUMBER(10)	INT
NCHAR_n	NCHAR(n)	CHAR(n)	CHAR(n)
NUMBER_n	NUMERIC(n)	NUMBER(n)	DECIMAL(n)
NVARCHAR_n	NVARCHAR(n)	VARCHAR2(n)	VARCHAR(n)
SMALLDATE	SMALLDATETIME	DATE	TIMESTAMP
SMALLINT	SMALLINT	SMALLINT	SMALLINT
TINYINT	TINYINT	SMALLINT	SMALLINT
VARCHAR_n	VARCHAR(n)	VARCHAR2(n)	VARCHAR(n)

Nvarchar and Nchar

The NVARCHAR and NCHAR data types can store UniCode (UCS-2) data on SQL Server data marts only. If you wish to include multiple language strings in your data mart, use this data type on SQL Server.

Oracle and DB-2 data marts support multiple language strings with UTF-8 data in VARCHAR and CHAR datatypes only.

If you import UCS-2-enabled data into a non-UCS-2 enabled EpiMart, or vice versa, you can use the **Adjust NVARCHARs** tool to set all data types to the appropriate settings. When you run this tool all physical type references to NVARCHAR_n or VARCHAR_n are changed to match the **External Interfaces > Meta > DBEncodingScheme** configuration variable.

The tool converts all data types, so it should not be used if mixed settings are necessary.

Note: NVARCHAR and NCHAR are translated by the tool to the same data types as VARCHAR and CHAR for Oracle and DB2 data marts.

BIGINT Data Types

DB2 and SQLServer provide the 8 byte integer type BIGINT. Using BIGINTs as sskeys for tables with row counts exceeding the capacity of 4 byte integers improves Infor Campaign Management's performance and reduces storage needs.

BIGINT data types are available for fact columns, fact sskeys, external table columns, dimension sskeys, dimension keys, degenerate dimension sskeys, and dimension columns.

Data Type Conversions

In some circumstances you can change the data types of fact, dimension, and external table columns without rebuilding the table. The following limitations apply:

- Physical type conversion (for example, from CHAR to VARCHAR) is not supported for Oracle or DB2 data marts. SQL Server physical type conversion is supported as outlined in the SQL Server documentation.
- A table column cannot be redefined to be smaller than its original size.
- Infor uses either `CREATE TABLE AS SELECT` statements (or the equivalent in SQL Server and DB2 databases) or `ALTER` statements in schema adaptation operations. Depending on the database and what it supports, the data type resulting from `CREATE TABLE AS SELECT` statements can not be an exact match with the data type defined in the EpiMeta database.

The following tables summarize data type attributes and legal conversion operations.

Table 101: Data Type Conversion Limitations 1

	CHAR	NCHAR	DECIMAL	FLOAT
Starting length	1	1	1	0
Length limit	255	255	30	0
Intmappable	Yes	Yes	No	No
Dimension column length increase (Adapt Schema Mode)	SQL Server and Oracle only.	Supported in all databases.	SQL Server only.	Not supported.
Intmapped dimension column length increase (Adapt Schema Mode)	SQL Server and Oracle only.	Supported in all databases.	Not supported.	Not supported.
SSKey length increase in intmapped table (Adapt Schema Mode)	SQL Server and Oracle only.	Supported in all databases.	Not supported.	Not supported.
Fact column length increase (Adapt Schema Mode)	SQL Server and Oracle only.	Supported in all databases.	SQL Server only.	Not supported.
Add a new column (Adapt Schema Mode)	Supported in all databases.	Supported in all databases.	SQL Server only.	SQL Server only.

Table 102: Data Type Conversion Limitations 2

	INT	NUMBER	NVARCHAR	VARCHAR
Starting length	0	1	1	0
Length limit	0	28	2000	2000
Intmappable	No	No	Yes (if length > 5)	Yes (if length > 5)
Dimension column length increase (Adapt Schema Mode)	Not supported.	SQL Server only.	Supported in all databases.	Supported in all databases.
Intmapped dimension column length increase (Adapt Schema Mode)	Not supported.	Not supported.	Supported in all databases.	Supported in all databases.

	INT	NUMBER	NVARCHAR	VARCHAR
SSKey length increase in intmapped table (Adapt Schema Mode)	Not supported.	Not supported.	Supported in all databases.	Supported in all databases.
Fact column length increase (Adapt Schema Mode)	Not supported.	SQL Server only.	Supported in all databases.	Supported in all databases.
Add a new column (Adapt Schema Mode)	SQL Server only.	SQL Server only.	Supported in all databases.	Supported in all databases.

Table 103: Data Type Conversion Limitations 3

	SMALLDATE	SMALLINT	TINYINT	BIGDATE
Starting length	0	0	0	0
Length limit	0	0	0	0
Intmappable	No	No	No	No
Dimension column length increase (Adapt Schema Mode)	Not supported.	Not supported.	Not supported.	Not supported.
Intmapped dimension column length increase (Adapt Schema Mode)	Not supported.	Not supported.	Not supported.	Not supported.
SSKey length increase in intmapped table (Adapt Schema Mode)	Not supported.	Not supported.	Not supported.	Not supported.
Fact column length increase (Adapt Schema Mode)	Not supported.	Not supported.	Not supported.	Not supported.
Add a new column (Adapt Schema Mode)	SQL Server only.	SQL Server only.	SQL Server only.	Not supported.

Table 104: Data Type Conversion Limitations 4

	BIGINT
Starting length	0
Length limit	0
Intmappable	No
Dimension column length increase (Adapt Schema Mode)	Not supported.
Intmapped dimension column length increase (Adapt Schema Mode)	Not supported.
SSKey length increase in intmapped table (Adapt Schema Mode)	Not supported.
Fact column length increase (Adapt Schema Mode)	Not supported.
Add a new column (Adapt Schema Mode)	SQL Server only.

Campaign-Related Dimensions

D

The Campaign, Cell, and Message dimensions have several built-in columns that cannot be removed or modified. These columns are found in both the dimension tables and the backfeed tables.

The Campaign Dimension

The Campaign dimension has the following built-in columns:

Table 105: Campaign Dimension Columns

Column	Description	Backfeed Contents
campaign_code	A unique identifying code for the campaign.	The backfeed table field that is used to identify a campaign for exclusion purposes. Campaign_code is assigned by the end user. If the end user does not specify a campaign code, it is generated when the campaign extraction job is run and is not displayed in the front end.
campaign_id	A unique identifier for the campaign run.	Automatically set to the same value as campaign_sskey, which is guaranteed to be unique for each campaign run.
campaign_label	A descriptive identifier for the campaign.	Assigned by the end user.
campaign_shared_id	An identifier for campaigns shared with Marketing Resource Management. Present only in a Marketing Resource Management-enabled system.	Automatically assigned.

Column	Description	Backfeed Contents
campaign_objective	A description of the goal or objective of the campaign.	Assigned by the end user.
campaign_ver	An identifying code for the campaign version.	Automatically incremented every time a campaign is saved.
createdatetime	The date and time that the campaign was first created.	Automatically assigned when the campaign is created.
def_cost_per_treatment	The default cost per treatment in the campaign.	Assigned by the end user.
def_profit_per_response	The default expected profit per response in the campaign.	Assigned by the end user.
def_response_rate	The default expected response rate for the campaign.	Assigned by the end user.
def_rev_per_response	The default expected revenue per response in the campaign.	Assigned by the end user.
export_time_s	The number of seconds taken for campaign export.	Automatically assigned during campaign export.
filename	The filename of the campaign in the Report Gallery.	Automatically assigned.
fixed_setup_cost	The total fixed setup cost of the campaign.	Assigned by the end user.
folder_path	The folder path of the campaign in the Report Gallery.	Automatically generated.
is_inference_campaign	This column has a value of 1 if this is an inference campaign and a value of 0 or NULL otherwise.	Automatically assigned during campaign export.
list_time_s	The number of seconds taken to generate the campaign list.	Automatically assigned during list generation.
num_exported_cells	The number of campaign cells exported.	Automatically assigned during campaign export.
num_exported_records	The total number of records exported.	Automatically assigned during campaign export.
saving_user	The Infor Omni-Channel Campaign Management user saved the campaign.	Automatically assigned when campaign is saved.
scheduling_user	The Infor Omni-Channel Campaign Management user scheduled the campaign.	Automatically assigned when campaign is scheduled.

Column	Description	Backfeed Contents
target_audience	A string description of the target audience for a campaign.	Assigned by the end user.
track_campaign_code	In an inference campaign, the campaign_code value of the campaign for which responses are being inferred.	Automatically assigned during campaign export.
campaign_ia_id	The campaign ID shared with Infor Interaction Advisor for inbound campaigns, if applicable.	Automatically assigned.
has_inbound_cells	Whether the campaign includes inbound cells	Automatically assigned.
has_outbound_cells	Whether the campaign includes outbound cells.	Automatically assigned.
program_budget	The budget of the program to which this campaign belongs.	Assigned by the end user for each program.
program_end	The end date of the program to which this campaign belongs.	Assigned by the end user for each program.
program_ID	The ID of the program to which this campaign belongs.	Automatically assigned for each program.
program_name	The name of the program to which this campaign belongs.	Assigned by the end user for each program.
program_start	The start date of the program to which this campaign belongs.	Assigned by the end user for each program.
program_type	The type of the program to which this campaign belongs.	Assigned by the end user for each program.

The Cell Dimension

The Cell dimension has the following built-in columns:

Table 106: Cell Dimension Columns

Column	Description	Backfeed Contents
campaign_id	The unique identifier for the associated campaign.	Automatically assigned the same value as campaign_sskey in the associated campaign.

Column	Description	Backfeed Contents
cell_id	A unique identifier for the cell run.	Automatically set to the same value as cell_sskey, which is formed by concatenating the associated campaign_sskey value, the cell node number and the channelkey. The cell node number uniquely identifies a cell within a campaign. The channel key for a control cell without any channel is not specified and is not added to the cell_id.
cell_label	A descriptive identifier for a cell.	Assigned by the end user.
cell_position	The position of the cell in the segmentation tree.	Automatically assigned.
cell_shared_id	An identifier for cells shared with Marketing Resource Management. Present only in a Marketing Resource Management-enabled system.	Automatically assigned.
cell_size	The number of campaign recipients in the cell.	Automatically assigned.
cell_ver	An identifying code for the cell and campaign version.	Automatically generated by concatenating the associated campaign_ver value with the cell node number.
est_cell_cost	The estimated cost for the cell.	Automatically assigned.
est_cell_profit	The estimated profit for the cell.	Automatically assigned.
est_cell_response_rate	The estimated response rate for the cell.	Can be assigned by the end user. If the end user does not specify a value, then the value of def_response_rate in the associated campaign is used.
est_cell_revenue	The estimated revenue rate for the cell.	Automatically assigned.
est_profit_per_response	The estimated profit per response for the cell.	Can be assigned by the end user. If the end user does not specify a value, then the value of def_profit_per_response in the associated campaign is used.
est_revenue_per_response	The estimated revenue per response for the cell.	Can be assigned by the user. If the end user does not specify a

Column	Description	Backfeed Contents
		value, then the value of def_rev_per_response in the associated campaign is used.
is_control	Specifies whether the cell must be treated as a control group in Campaign Analysis. Control cells are used as baseline comparisons against cells with which the control cell is associated.	Assigned by the end user.
is_tracking_cell	This column has a value of 1 if the cell is used for tracking campaign output that has not yet been generated.	Automatically generated.
output_channel_label	The label of the output channel associated with the cell.	Automatically assigned.
output_file_name	The name of the output file associated with the cell.	Assigned by the end user if multiple output files are scheduled for a campaign.
output_format_label	The label of the output format associated with the cell.	Automatically assigned.
output_processor_label	The label of the output processor associated with the cell.	Automatically assigned.
segment_level_ n	The name of the nth segment filter level, where n can be any level between 1 and 10.	Automatically assigned based on the campaign segmentation hierarchy.
touchpoint	The name of the touchpoint that is associated with the cell.	Assigned by the end user.
treatment_code	A unique identifying code for the cell.	Usually assigned by the end user. If the end user does not specify a value, then the value 'N/A' is used. This is the backfeed table field that is used to identify a cell for exclusion purposes.
treatment_description	A description of the treatment used within the cell.	Assigned by the end user.
treatment_etc_fullpath	The full file path of the email treatment content.	Automatically assigned based on the user-selected email treatment content.
treatment_etc_name	The user name from the e-mail treatment content file.	Automatically assigned based on the user-selected email treatment content.

Column	Description	Backfeed Contents
treatment_shared_id	An identifier for treatments shared with Marketing Resource Management. Present only in a Marketing Resource Management-enabled system.	Automatically assigned.
treatment_unit_cost	The cost of a single treatment in the cell.	Can be assigned by the end user. If the end user does not specify a value, then the value of def_cost_per_treatment in the associated campaign is used.
cell_channel	The name of the channel used for this cell.	Automatically assigned based on selected channel.
cell_node_id	The ID of the campaign node for this cell.	Automatically assigned.
communication_id	The Infor Interaction Advisor ID associated with the communication used for this cell, if applicable.	Automatically assigned.
communication_name	The name of the Communication used for this cell.	Automatically assigned based on selected communication.
est_cell_benefit	The estimated benefit per response for this cell.	Assigned by the end user.
est_cell_cost_capacity	The estimated cost capacity for this cell.	Assigned by the end user.
est_cell_cost_per_response	The estimated cost per response for this cell.	Assigned by the end user.
est_cell_margin	The estimated margin for this cell.	Assigned by the end user.
is_inbound	Whether this is an inbound cell.	Automatically assigned.

The Message Dimension

The Message dimension has the following built-in columns:

Table 107: Cell Dimension Columns

Column	Description	Backfeed Contents
campaign_id	The unique identifier for the associated campaign.	Automatically assigned the same value as campaign_sskey in the associated campaign.
cost_per_message	The estimated cost of the message.	Assigned by the end user.
description	A description of the message.	Assigned by the end user.
est_profit_per_response	The estimated profit per response for the message.	Assigned by the end user.
est_revenue_per_response	The estimated revenue gained for each positive response.	Assigned by the end user.
estimated_response_rate	The estimated response rate for the message.	Assigned by the end user.
message_channel	The touchpoint used to deliver the message.	Assigned by the end user.
message_code	A unique code that identifies the message.	Assigned by the end user.

Object Types and Views

E

Note: Do not copy and paste scripts, commands or code from this document. Line breaks and some other characters picked up when you copy from the PDF can cause errors in the pasted text.

Infor Campaign Management allows you to control the physical properties of objects in your EpiCenter. This appendix lists the types of objects that can be configured in your EpiMeta, EpiMart, and EpiOp databases, along with their default tablespaces, naming conventions, and views.

Note: If a table is created in a system-managed tablespace on a DB2 database server, then indexes for that table must be created in the same tablespace.

Note: On an Oracle database server, the default tablespaces are created automatically by the database initialization scripts, as described in the *Infor Campaign Management Installation Guide*.

EpiMeta Object Types

The "Table, EpiMeta Object Types" on page 517 lists the configurable object types in an EpiMeta database. "Naming Conventions Legend" on page 531 provides a legend for codes that appear in the **Naming Conventions** column.

Table 108: EpiMeta Object Types

Object Type Code	Description	Object Naming Convention
META_DATA	EpiMeta metadata tables.	Various
META_IDX	EpiMeta metadata indexes.	Various
META_EPIAPP_DATA	EpiMeta temporary and scratch tables.	Various
META_EPIAPP_IDX	EpiMeta temporary and scratch indexes.	Various

EpiMart Object Types

The tables in this section list the configurable object types in an EpiMart database. "Table 109: EpiMart Object Types" on page 518 lists standard EpiMart objects, while "Table, DB2 Replication Object Types" on page 521 lists replication object types for DB2 and "Table, Partition Object Types" on page 522 lists table partition overrides for Oracle data marts.

The table below provides a legend for codes that appear in the Naming Conventions column.

Table 109: EpiMart Object Types

Object Type Code	Description	Object Naming Convention
DIM_AGG_DATA	Dimension aggregate tables.	• <dim>_<aaa>_<A|B> • <dim>_<aaa>_<A|B>_d • <dim>_<aaa>_<A|B>_p where # > 0
DIM_AGG_IDX	Indexes for dimension aggregate tables.	See "Index Naming Conventions" on page 526.
DIM_DATA	Base and thin dimension tables.	• <dim>_0_<A|B> • <dim>_t_<A|B>
DIM_IDX	Indexes for base and thin dimension tables.	See "Index Naming Conventions" on page 526.
DIM_MAP_DATA	Dimension mapping tables.	<dim>map_<A|B>
DIM_MAP_IDX	Indexes for dimension mapping tables.	See "Index Naming Conventions" on page 526.
DIM_FMAP_DATA	Dimension fusion mapping tables.	• <dim>fstage • <dim>fmap_<A|B> • <dim>fold_<A|B>
DIM_FMAP_IDX	Indexes for dimension fusion mapping tables.	See "Index Naming Conventions" on page 526.
DIM_STAGE_DATA	Dimension staging and semantic temporary tables.	<dim>stage
DIM_STAGE_IDX	Indexes for dimension staging and semantic temporary tables.	See "Index Naming Conventions" on page 526.
DIM_STAGE_DRV_DATA	Dimension-staging derived tables, such as the insert (ins) and update (upd) tables used for incremental updates.	• <dim>ins_<A|B> • <dim>upd_<A|B> • <dim>old_<A|B> • <dim>fold_<A|B>

Object Type Code	Description	Object Naming Convention
DIM_STAGE_DRV_IDX	Indexes for dimension-staging derived tables, such as the insert (ins) and update (upd) tables used for incremental updates.	See "Index Naming Conventions" on page 526.
EXT_DATA	External tables.	User configurable
EXT_IDX	Indexes for external tables.	User configurable
FACT_AGG_DATA	Fact aggregate tables.	• <fact>_<aaa>_<A|B> • <fact>_<aaa>_<A|B>_d • <fact>_<A|B>_SUSP • <dim13>_<DD>_<fact-Type>_<aaa>_<A|B> • <dim13>_<DD>_<fact-Type>_<aaa>_<A|B>_d
FACT_AGG_IDX	Indexes for fact aggregate tables.	See "Index Naming Conventions" on page 526.
FACT_DATA	Base fact tables (A and B tables).	• <fact>_0_<A|B> • <dim13>_<DD>_<fact-Type>_0_<A|B>
FACT_IDX	Indexes for base fact tables (A and B tables).	See "Index Naming Conventions" on page 526.
FACT_STAGE_DATA	Fact staging tables and semantic temporary tables.	• <fact>stage • <dim13>_<DD>_<fact-Type>stage
FACT_STAGE_IDX	Indexes for fact staging tables and semantic temporary tables.	See "Index Naming Conventions" on page 526.
FACT_STAGE_DRV_DATA	Fact-staging derived tables, such as the insert (ins) tables used in incremental updates.	• <fact>ins_<A|B> • <dim13>_<DD>_<fact-Type>ins_<A|B>
FACT_STAGE_DRV_IDX	Indexes for fact-staging derived tables, such as the insert (ins) tables used in incremental updates.	See "Index Naming Conventions" on page 526.
INTMAP_DATA	Dimension integer map tables.	IIMMAAPP_<iiii>
INTMAP_IDX	Dimension integer map indexes.	See "Index Naming Conventions" on page 526.
MART_EPIAPP_DATA	EpiMart temporary tables. Used primarily by the Infor Campaign	TTMMPP_<tttttt>_<nn>

Object Type Code	Description	Object Naming Convention
	Management Server. See "Temporary (TTMMPP) Tables" on page 524 for details.	
MART_EPIAPP_IDX	Indexes for EpiMart temporary tables. Used primarily by the Appserver.	Various.
MART_LIST_DATA	Saved and scored list tables (epikey based).	list_<LLLL>
MART_LIST_IDX	Indexes for Saved and scored list tables (epikey based).	See "Index Naming Conventions" on page 526.
MART_META_DATA	EpiMart metadata tables.	Various
MART_META_IDX	Indexes for EpiMart metadata tables.	Various
MOM_DIM_DATA	Momentum all/alns tables	• <dim>_all_<A|B> • <dim>_allns_<A|B>
MOM_DIM_IDX	Indexes for the Momentum all/alns tables	See "Index Naming Conventions" on page 526.
MOM_DIM_SAMPLE_DATA	Dimension sampling tables.	• <dim>_<DD><S>_<A|B> • <dim>_t<DD><S>_<A|B> • <dim>_all<S>_<A|B> • <dim>_alns<S>_<A|B>
MOM_DIM_SAMPLE_IDX	Indexes for dimension sampling tables.	See "Index Naming Conventions" on page 526.
MOM_FACT_DATA	Relation fact tables.	<rel>_r_<A|B>
MOM_FACT_IDX	Relation fact indexes.	See "Index Naming Conventions" on page 526.
MOM_FACT_SAMPLE_DATA	Fact and momentum relation sample tables.	• <fact>_<DD><S>_<A|B> • <dim13>_<DD>_<factType>_<DD><S>_<A|B> • <rel>_<DD><S>_<A|B>
MOM_FACT_SAMPLE_IDX	Indexes for fact and momentum relation sample tables.	See "Index Naming Conventions" on page 526.

DB2 Replication Object Types

The "Table, DB2 Replication Object Types" on page 521 lists the configurable object types for replication in DB2.

Table 110: DB2 Replication Object Types

Object Type Code	Description	Object Naming Convention
DIM_REPL_DATA	Replicated base and thin dimension tables.	• <dim>_0_RP_<A|B> • <dim>_t_RP_<A|B>
DIM_REPL_IDX	Indexes for replicated base and thin dimension tables.	See "Index Naming Conventions" on page 526.
DIM_MAP_REPL_DATA	Replicated dimension mapping tables.	<dim>map_RP_<A|B>
DIM_MAP_REPL_IDX	Indexes for replicated dimension mapping tables.	See "Index Naming Conventions" on page 526.
FACT_REPL_DATA	Replicated base fact tables (A and B tables).	• <fact>_0_RP_<A|B> • <dim13>_<DD>_<factType>_0_RP_<A|B>
FACT_REPL_IDX	Indexes for replicated base fact tables (A and B tables).	See "Index Naming Conventions" on page 526.
INTMAP_REPL_DATA	Replicated dimension integer map tables.	IIMMAAPP_RP_<iiii>
INTMAP_REPL_IDX	Replicated dimension integer map indexes.	See "Index Naming Conventions" on page 526.
MART_EPIAPP_REPL_DATA	Replicated EpiMart temporary tables. Used primarily by the Infor Campaign Management Server. See "Temporary (TTMMPP) Tables" on page 524 for details.	TTMMPP_<tttttt>_<nn>
MART_EPIAPP_REPL_IDX	Indexes for replicated EpiMart temporary tables. Used primarily by the Appserver.	Various.

Partition Object Types

The "Table, Partition Object Types" on page 522 lists the configurable object types for fact and dimension partition-level overrides on Oracle.

Table 111: Partition Object Types

Object Type Code	Description	Partition Naming Convention
DIM_PART_DATA	Partition-level object property overrides for base- and thin-partitioned EpiMart dimension tables.	<dim><partition_list_order>, where partition_list_order is the number listorder as specified in metadata.
DIM_PART_IDX	Indexes for partition-level object property overrides for base- and thin-partitioned EpiMart dimension tables.	<dim_index><partition_list_order>, where partition_list_order is the number listorder as specified in metadata.
FACT_PART_DATA	Table partition-level object property overrides for base partitioned EpiMart fact tables	<fact><partition_list_order>, where partition_list_order is the number listorder as specified in metadata.
FACT_PART_IDX	Indexes for table partition-level object property overrides for base partitioned EpiMart fact tables.	<fact_index><partition_list_order>, where partition_list_order is the number listorder as specified in metadata.
FACT_PART_STAGE_DRV_DATA	Table partition-level object property overrides for partitioned EpiMart fact stage derived tables.	<fact><partition_list_order>, where partition_list_order is the number listorder as specified in metadata.
FACT_PART_STAGE_DRV_IDX	Index partition-level object property overrides for partitioned EpiMart fact stage derived tables.	<fact_index><partition_list_order>, where partition_list_order is the number listorder as specified in metadata.

Physical Object Types

The "Table, Physical Object Types" on page 522 lists the physical object types that determine in which tablespace Infor Campaign Management creates a table in Oracle and DB2.

Table 112: Physical Object Types

Object Type Code	Tablespace Naming Convention	Description
FACT_STAGE_DATA	<fact>_0_MFL	Temporary table created by semantics and only present while a semantic is running. (Infor Campaign Management does not drop the table, however, if the semantic is run in debug mode.)

Object Type Code	Tablespace Naming Convention	Description
FACT_STAGE_DATA	<fact>STAGE_DST	Temporary table created by semantics and only present while a semantic is running. (Infor Campaign Management does not drop the table, however, if the semantic is run in debug mode.)
FACT_AGG_DATA	<fact>_<aaa>_<A|B>	
FACT_AGG_DATA	<fact>_<A|B>_SUSP	Table used for aggregate processing.

EpiOp Object Types

The "Table, EpiOp Object Types" on page 523 lists t he configurable object types in an EpiOp database. "Naming Conventions Legend" on page 531 provides a legend for codes that appear in the `Naming Conventions` column.

Table 113: EpiOp Object Types

Object Type Code	Description	Object Naming Convention
BF_DIM_CURR_DATA	Infor Omni-Channel Campaign Management P/Q dimension backfeed tables.	<dim>_bf_<P|Q>
BF_DIM_CURR_IDX	Indexes for Infor Omni-Channel Campaign Management P/Q dimension backfeed tables.	See "Index Naming Conventions" on page 526.
BF_DIM_HIST_DATA	Infor Omni-Channel Campaign Management A/B dimension backfeed tables.	<dim>_bf_<A|B>
BF_DIM_HIST_IDX	Indexes for Infor Omni-Channel Campaign Management A/B dimension backfeed tables.	See "Index Naming Conventions" on page 526.
BF_FACT_CURR_DATA	Infor Omni-Channel Campaign Management P/Q fact backfeed tables.	• <dim13>_<DD>_<com|msg>_bf_<P|Q> • <dm13>_<DD>_<com|msg>_bf_<AB|PQ>
BF_FACT_CURR_IDX	Indexes for Infor Omni-Channel Campaign Management P/Q fact backfeed tables.	See "Index Naming Conventions" on page 526.

Object Type Code	Description	Object Naming Convention
BF_FACT_HIST_DATA	Infor Omni-Channel Campaign Management A/B fact backfeed tables.	<dim13>_<DD>_<com|msg>_bf_<A|B>
BF_FACT_HIST_IDX	Indexes for Infor Omni-Channel Campaign Management A/B fact backfeed tables.	See "Index Naming Conventions" on page 526.
EPILOG_DATA	Query and extraction log tables.	Various
EPILOG_IDX	Query and extraction log indexes.	Various
OP_EPIAPP_DATA	EpiOp temporary tables. See Also, "Temporary (TTMMPP) Tables" on page 524 for details.	TTMMPP_<tttttt>_<nn>
OP_EPIAPP_IDX	EpiOp temporary indexes.	Various.
OP_LIST_DATA	Saved and scored list tables (sskey based) and campaign export tables (as listed in the campaign export header file).	List tables: list_<LLLL>, Campaign Export tables: EMXP_<campaign_sskey>_<output_file_listorder>_<placement_listorder>
OP_LIST_IDX	Indexes for saved and scored list tables (sskey based) and campaign export tables.	See "Index Naming Conventions" on page 526.
OP_META_DATA	EpiOp metadata tables.	Various
OP_META_IDX	EpiOp metadata indexes.	Various

Temporary (TTMMPP) Tables

The Infor Campaign Management Server names temporary (temp) files using the following convention:

TTMMPP_<tttttt>_<nn>

where

- tttttt is the same for all tables created for a query and is distinct to that query
- nn is a sequential number starting at one (1) and incremented by one for each table created in that query

The Infor Campaign Management Server uses this naming convention for all temporary tables, including query-engine temporary tables. For example, if you run a campaign:

- tttttt is the internally generated number corresponding to the campaign
- nn is an incrementing list of the temporary tables for that campaign

Temporary Table Storage

The query engine automatically creates internal temporary tables as needed. This is automatic and a user cannot control the creation or cleanup of internal temporary tables. The Infor Campaign Management Server drops most of these temporary tables when it no longer needs them, but keeps those that it can use for computations and those which can be reused.

The Infor Campaign Management Server stores these temporary tables in one of the following caches:

- Query cache
- Sub-query cache
- Momentum cache

The Infor Campaign Management Server stores these caches in the tablespace set aside for the following object types:

- MART_EPIAPP_DATA object type (EpiMart)
- OP_EPIAPP_DATA object type (EpiOp)

Admin Manager administrators can set the tablespace for some object types through the **EpiCenter > Physical Object Properties** dialog box.

Temporary Table Cleanup

The Infor Campaign Management Server drops many of the temporary tables when it is refreshed. For some, Infor Campaign Management Server deletes the temporary table based on the value of TempTableExpirationTime, set in the following location in Admin Manager:

Configuration > All Settings > Optimization > Query

Typically, this value is set to three days.

Occasionally, a temporary tables becomes an orphan—a temporary table that is not automatically cleaned up when no longer needed—for a variety of reasons. (One way to “orphan” a table is to stop an extraction job in the middle of its execution.) In this case, you can use the Purge utility (**EpiCenter > Purge** to remove orphaned temporary tables. (See Also, "Purging EpiMart and EpiOp Tables" on page 375.)

The EpiManager Purge utility displays only temporary list tables as available for purging.

- Permanent lists are materialized lists deliberately saved through the Report Gallery. Permanent lists are first saved in the EpiMart, then later flushed to the EpiOp by various actions.
- Temporary list tables—the names of which look exactly like those for permanent tables—are not flushed to the EpiOp, and they linger in the EpiMart until deliberately purged using EpiManager. Infor Campaign Management creates temporary list tables when saving the settings for a list, downloading a list for viewing, and similar actions. These are list tables which is not saved as materialized lists and only need to be used temporarily.

If a list is a permanent list, then its ID must exist in one of the following tables:

- epifile_list

- Contains the list table IDs of the permanent list tables in EpiOp, which are not candidates for purge.
- report_server_list
- Contains the list table IDs of the permanent list tables in the EpiMart, which are not candidates for purge.

The converse is equally true. If the ID for a list does not exist in one of these tables, then it is a temporary list and it is available for purging.

Automatic Cache Purge after Application Server Restart

No action is taken when the Infor Campaign Management Server is stopped. However, when it is restarted, Infor Campaign Management Server clears its caches, meaning that the memory is simply cleared and cached tables are no longer recognized. After restart of the Infor Campaign Management Server, any cached tables that previously appeared in the Infor Campaign Management Monitor's Query cache list are no longer shown. (The Infor Campaign Management Monitor—only available to those with administrative privileges in the front-end Infor Campaign Management interface—displays a Query cache which contains the results of the most recent queries that users have run.)

TTMMPP tables, however, remain in the EpiMart as "dead" items. These TTMMPP tables are cleared through one of the following mechanisms:

- by the Infor Campaign Management Server, which clears them after three days:
- Using a similar concept as with list tables, the IDs of the TTMMPP tables appear in a table called temp_tbl. The Infor Campaign Management Server deletes those temporary tables for which the entry in temp_tbl is not updated for at least three days.
- by manual purging using Admin Manager Purge utility:
- The Purge utility displays only the older TTMMPP tables as purge candidates.

There is no explicit flushing of lists from the mart(s) to the EpiOp when the Infor Campaign Management Server is restarted. However, the periodic flushing mechanism built into Infor Campaign Management Server is reactivated whenever the Infor Campaign Management Server is restarted.

Index Naming Conventions

Indexes are named with the name of the base table that it references plus a 2- or 3-character index code to identify the type of index. For example:

```
<Mart/Op actual database table name>_<index suffix>
```

The "Table, Valid Index Suffixes" on page 527 lists valid index suffixes.

Table 114: Valid Index Suffixes

Suffix Code	Description
PK	Primary Key / Unique index
AK	Alternate Key index
I<n>	Non-primary key, non-alternate key implicit indexes, where <n> is the numeric identifier for the implicit index.
<xx>	Explicit indexes, where <xx> is the numeric identifier for the explicit index.
Sxx	Shadow indexes (real key columns instead of dimension key columns) for fact custom indexes.
<DD>	Dimension-role-specific index. For example, relation tables have single-column indexes on each dimension role included in the table. The index suffix is the 2-character dimension base abbreviation for the dimension role that is being indexed.

Views

The Infor Campaign Management application only creates views on the current tables for selected base objects. These views are summarized in the following tables. "Naming Conventions Legend" on page 531provides a legend for codes that appear in the Naming Conventions column.

Note: The Create Current Views extraction step fails for a given view if the view is in use.

Fact and Dimension Table Views

The following views display information about fact and dimension tables in your EpiMart.

Table 115: Table View Descriptions

Description	Naming Standard
“Raw” non-communication/message base tables. This view does not convert intmapped columns to their actual strings.	<dim | fact>_0_RV
Non-communication/message base tables. This view converts intmapped columns to their actual strings.	<dim | fact>_0_V

Description	Naming Standard
Non-communication/message base tables. This view converts intmapped columns to their actual strings and includes sskeys.	<dim | fact>_0_SV
“Raw” communication base tables. This view does not convert intmapped columns to their actual strings.	<dim13>_<DD>_com_0_RV
Communication base tables. This view converts intmapped columns to their actual strings.	<dim13>_<DD>_com_0_V
Communication base tables. This view converts intmapped columns to their actual strings and includes sskeys.	<dim13>_<DD>_com_0_SV
“Raw” message base tables. This view does not convert intmapped columns to their actual strings.	<dim13>_<DD>_msg_0_RV
Message base tables. This view converts intmapped columns to their actual strings.	<dim13>_<DD>_msg_0_V
Message base tables. This view converts intmapped columns to their actual strings and includes sskeys.	<dim13>_<DD>_msg_0_SV
Dimension mapping table views	<dim>MAP_RV
Integer mapping column views. These columns show the integer mapping of actual strings for the current set of mirrored tables.	<dim11>_<intmap_col_11>[<iiii>]_CV
Backfeed dimension table views	<dim>_BF_V
Backfeed dimension table views with sskeys	<dim>_BF_SV
Backfeed communication table views	<dim13>_<DD>_com_BF_V
Backfeed communication table views with sskeys	<dim13>_<DD>_com_BF_SV
Backfeed message table views	<dim13>_<DD>_msg_BF_V
Backfeed message table views with sskeys	<dim13>_<DD>_msg_BF_SV

Built-Object Views

The following additional views provide information on built objects in your EpiMart and EpiOp databases. Mart views are stored in your EpiMeta, and Op views are stored in your EpiOp. Note the following:

- `*_tbl_*` views contain tables.
- `*_idx_*` views contain indexes.

Table 116: EpiMart/Op Built-in Object View Names

EpiMart Views	Description
built_all_demo_tbl_view built_all_demo_idx_view	All demographic tables and sample tables.
built_dim_agg_tbl_view built_dim_agg_idx_view	All dimension aggregates.
built_dim_sample_tbl_view built_dim_sample_idx_view	All dimension sample objects and thin sample objects.
built_fact_agg_tbl_view built_fact_agg_idx_view	All fact aggregates.
built_fact_sample_tbl_view built_fact_sample_idx_view	All fact sample objects and thin sample objects
built_intmap_tbl_view built_intmap_idx_view	All dimension integer maps.
built_keymap_tbl_view built_keymap_idx_view	Aggregate keymap objects.
built_many_sample_tbl_view built_many_sample_idx_view	Relation many sample objects, and relation many sample thin objects. These tables describe one-to-many relationships between tables - for example, the if you have Group and Individual dimensions, the individuals who belong to groups are tracked in many sample tables.
built_mart_list_tbl_view built_mart_list_idx_view	Lists the objects in the EpiMart.
built_mart_schema_tbl_view built_mart_schema_idx_view	All of the fact/dim/external base objects, including base and thin dimension tables.
built_op_list_tbl_view built_op_list_idx_view	Lists all of the objects in the EpiOp.
built_relation_tbl_view built_relation_idx_view	Relation objects and relation base sample objects. These are relation objects and sample objects built by MomBuilder for relations defined in schema.

EpiMart Views	Description
EpiOp Views	Description
built_op_backfeed_tbl_view built_op_backfeed_idx_view	All backfeed objects, including current, history and insert tables for facts and dimensions.
built_cm_exp_tbl_view	Built campaign export tables. Contains P/Q back-feed tables.

View Columns

The "Table, View Column Descriptions" on page 530 describes the columns that are included with each view.

Table 117: View Column Descriptions

Column Name	Description
built_tbl_name	The built table name (without mirror letter).
logical_tbl_name	The logical parent table name (For example, for a fact built derived table, this column contains the fact base table name).
logical_subobject_id	For internal use.
data mart_letter	The mirror letter. This value is null if not mirrored, or A, B, P, or Q, depending on the object.
attach_key	For internal use.
base_key	For internal use.
built_tbl_type	The built table type. This column categorizes built table objects at one level below physical object type. (For example, FactIns, DimensionIns, and so on.)
tbl_phys_obj_type	The physcial object type that is used for data ob-jects.
idx_phys_obj_type	The physical object type that is used for related index objects.
table_size	Reserved for future use.
index_suffix	(Index views only) The index suffix. See "Index Naming Conventions" on page 526.
bitmap_flag	(Index views only) Specifies whether this is a bitmap index (Oracle only).

Column Name	Description
cluster_flag	(Index views only) Specifies whether this is a clustered index (SQL Server only).
unique_flag	(Index views only) Specifies whether this is a unique index.
imp_idx_flag	(Index views only) Specifies whether this is an implicit index built by generate schema or semantics, as opposed to a MomBuilder accelerator.

Naming Conventions Legend

The "Table, Naming Convention Legend" on page 531 provides a legend for symbols in the **Naming Convention** column of tables that appear within this appendix.

Table 118: Naming Convention Legend

Code	Refers to	Maximum Length
<dim>	Dimension base table name	20
<dim13>	First 13 characters of the dimension base table name	13
<fact>	Fact base table name	20
<rel>	Relation base table name	20
<DD>	Dimension base table abbreviation	2
<aaa>	Aggregate number (0 refers to the base table. All other numbers must be greater than zero.)	3
<S>	Sampling ratio (for example, 2 = 1/10^2)	1
<A|B><A|B|P|Q>	Mirror letter (see "Data Mart Mirroring: A and B Tables" on page 191).	1
<iiii>	Integer map id, as defined in dim_col_intmap.unique_id	4
<LLLL>	List id as defined in epi-file_list.list_id / report_server_list.list_id	4

Code	Refers to	Maximum Length
<tttttt>	Temp table key, as defined in temp_tbl.temp_tbl_key	6
<nn>	Temp table seq key, determined by Infor Campaign Management Server.	2
<factType>	The type of list-producing dimension fact. Values include: • com. A communication fact table. There is one communication fact table for each list-producing base dimension that is marked as Backfed. • msg. A message fact table. There is one message fact table for each list-producing base dimension that is marked as Backfed. • ifr. An inferred response fact table. There is one inferred response fact table for each list-producing base dimension that is marked for Inferred Response. • sed. A seed fact table. There is one seed fact table for each list-producing base dimension that is marked for Seeds.	n/a

Components of Built-in Semantics

F

Most of the pre-defined Infor Campaign Management semantic types are based on the options described in "Components of Dimension Semantics" on page 225, and "Components of Fact Semantics" on page 239. This appendix lists the options that are used for all built-in semantic types that are based on these options.

Built-in Fact Semantics

"Table: Fact Semantic Types, on page F-2" on page 533, "Table: Fact Semantic Types, on page F-2" on page 534, "Table: Fact Semantic Types, on page F-3" on page 534, and "Table: Fact Semantic Types, on page F-4" on page 535 show the options that are used for pre-defined fact semantic types.

Note: The Count Unjoined, First/Last Fact, and Reload Date Fact semantic types are not based on these options. These semantic types use custom templates to implement their special logic.

Table 119: Fact Semantic Types

	Initial Load Fact	Initial Load Fact, Truncate Current	Initial Load Statelike	Initial Load Transactional/ Statelike
Earlier Date in Stage/ Current	Error	Error	Current	Current
Force Close	No	No	No	No
Later Date in Stage	All	All	All	All
Later Date in Stage/ Current	Error	Error	Both	Both
Row type	Transactional	Transactional	Statelike	Transactional/ Statelike
Same Date in Stage	All	All	All	All

	Initial Load Fact	Initial Load Fact, Truncate Current	Initial Load Statelike	Initial Load Transactional/ Statelike
Same Date in Stage/ Current	Error	Error	Both	Both
Truncate Current Fact	No	Yes	No	No
Use Current	No	No	No	No

Table 120: Fact Semantic Types

	Initial Load Transactional/ Statelike/ Force Close	Pipelined	Pipelined/Forceclose	Statelike
Earlier Date in Stage/ Current	Current	Current	Current	Current
Force Close	Yes	No	Yes	No
Later Date in Stage	All	All	All	All
Later Date in Stage/ Current	Both	Both	Both	Both
Row type	Transactional/ Statelike	Pipelined	Pipelined	Statelike
Same Date in Stage	All	All	All	All
Same Date in Stage/ Current	Both	Both	Both	Both
Truncate Current Fact	No	No	No	No
Use Current	No	Yes	Yes	Yes

Table 121: Fact Semantic Types

	Statelike/ Error on Stage Duplicates	Statelike/ Single Delta	Streaming Fact	Transactional Incremental Streaming
Earlier Date in Stage/ Current	Current	Current	Both	Both
Force Close	No	No	No	No
Later Date in Stage	Error	First	All	All

	Statelike/ Error on Stage Duplicates	Statelike/ Single Delta	Streaming Fact	Transactional Incremental Streaming
Later Date in Stage/ Current	Both	Both	Both	Both
Row type	Statelike	Statelike	Streaming Facts	Streaming Facts
Same Date in Stage	Error	All	All	All
Same Date in Stage/ Current	Both	Both	Both	Both
Truncate Current Fact	No	No	No	No
Use Current	Yes	Yes	No	Yes

Table 122: Fact Semantic Types

	Transactional	Transactional/ Dedup Stage	Transactional/ Statelike
Earlier Date in Stage/ Current	Current	Current	Current
Force Close	No	No	No
Later Date in Stage	All	First	All
Later Date in Stage/ Current	Both	Both	Both
Row type	Transactional	Transactional	Transactional/ Statelike
Same Date in Stage	All	First	All
Same Date in Stage/ Current	Current	Current	Both
Truncate Current Fact	No	No	No
Use Current	Yes	Yes	Yes

Table 123: Fact Semantic Types

	Transactional/ Statelike/ Force Close
Earlier Date in Stage/ Current	Current
Force Close	Yes
Later Date in Stage	All

	Transactional/ Statelike/ Force Close
Later Date in Stage/ Current	Both
Row type	Transactional/ Statelike
Same Date in Stage	All
Same Date in Stage/ Current	Both
Truncate Current Fact	No
Use Current	Yes

Built-in Dimension Semantics

"Table: Dimension Semantic Types, on page F-5" on page 536, "Table: Dimension Semantic Types, on page F-5" on page 537, and "Table: Dimension Semantic Types, on page F-6" on page 537 show the options that are used for pre-defined dimension semantic types.

Note: The Backfeed Dimension, Latest Dimension Value with Fusion/Fission, and Latest Dimension Value, Preserve Fusion semantic types are not based on these options. These semantic types use custom templates to implement their special logic.

Table 124: Dimension Semantic Types

	First Dimension Value	Initial Load Dimension	Initial Load Dimension, Truncate Current	Latest Dimension Value
Base Updates	No Update	No Update	No Update	Any Update
New Rows	New SSKey	Stage Initial	Stage Initial	New SSKey
Rewrite Rows	No Rewrite	No Rewrite	No Rewrite	No Rewrite
Stage Duplicates	De-Dup	De-Dup	De-Dup	De-Dup
Truncate Current	No	No	Yes	No
Update Criteria	Always	Always	Always	Always

Table 125: Dimension Semantic Types

	Latest Dimension Value, Ignore Unknowns	Latest Dimension Value, Rewrite All	Slowly Changing Dimensions	Slowly Changing Dimensions, Ignore Unknowns
Base Updates	Any Update	Any Update	No Update	No Update
New Rows	New SSKey	New SSKey	Any Update	Any Update
Rewrite Rows	No Rewrite	Rewrite All	No Rewrite	No Rewrite
Stage Duplicates	De-Dup	De-Dup	De-Dup	De-Dup
Truncate Current	No	No	No	No
Update Criteria	Ignore Unknown	Always	Always	Ignore Unknown

Table 126: Dimension Semantic Types

	Slowly Changing/ First Dim Value, Ignore Unknown	Slowly Changing Primary, Latest Secondary	Streaming Dimension
Base Updates	No Update	Secondary Update	No Update
New Rows	Primary Update	Primary Update	Dimension Streaming
Rewrite Rows	No Rewrite	No Rewrite	No Rewrite
Stage Duplicates	De-Dup	De-Dup	De-Dup
Truncate Current	No	No	No
Update Criteria	Ignore Unknown	Always	Always

Job Validation Checks

G

This appendix describes some job validation errors that can result from misconfigured extraction jobs.

When you run an extraction job, EpiChannel runs job validation against the job structure to verify that the job is configured correctly. Incorrect job configuration can result in inconsistent data in your EpiCenter. If the job validation checks identify a serious configuration problem, EpiChannel returns an error.

Job validation errors are prefaced by an extraction exception. These exceptions all belong to the following class:

```
com.epiphany.etl.exception.EtlException
```

The errors in this appendix are listed by exception name, and grouped into errors, which prevent the job from running, and warnings, which do not.

Errors

Table 127: Job Validation Errors

Exception	Test Condition
DimInitSemanticInitExtract	Checks that initial semantics are only run against dimensions that are initial-loaded in the job. A dimension must have an initial extraction if an initial semantic is used, and only initial semantics if an initial extraction is performed.
DimRoleSemanticFactExtractError	If a fact has a dimension role where the dimension is mapped, enabled for memory lookup, and extracted in this job, then dimension semantics must be run on the dimension prior to fact extraction.

Exception	Test Condition
DuplicateSemantics	Checks to ensure that no more than one semantic is applied per fact or dimension per job (with the exception of fact-combining or non-state-altering semantics.)
ExtractionAfterSemantic	Extraction for a fact or dimension must never follow a semantic for that fact or dimension (unless the semantic is not state-altering), unless there is an intervening commit step.
ExtractionStepNotEnabled	Checks whether all extraction steps in a job are valid. This is triggered if a step is not enabled in the global object gallery. Enable the global extraction step in the Object Gallery by selecting it, checking the Valid option (in the right pane of the Job dialog box), and running Scrutiny. You can also disable the corresponding job step (in the left pane of the Job dialog box, Extraction Steps tab), or remove the step from the job.
FactStreamingDimensionNotLookup	If a fact is streaming extracted in a job, all associated dimensions must be fact-streaming or unmapped.
FusionDimSemanticNoFusionFactSemantic	This rule checks that fusion semantics are applied consistently. It checks each of the following conditions: • If you have a fact table to which a fusion semantic is applied, then it must include at least one fusion-enabled dimension in its dimensionality • If a fusion semantic is applied to a dimension table, then a fusion-enabled fact semantic instance must be applied to every fact that joins the table, and MomBuilder must be run in Rebuild mode. • If a fact does not have a fusion semantic, it cannot have any associated dimensions with fusion semantics.
HistoryRewriteDimensionRequiresFact	If a history-rewriting semantic is run on a dimension on which a fact has an aggregate, and AggBuilder is run during the job, then a semantic must be run on that fact during the job.

Exception	Test Condition
InitialMapDimFactError	If a dimension is mapped and is initial loaded, all facts referencing this dimension must either be: • truncated, initial loaded, and have an initial semantic, or • truncated, followed by an initial semantic.
InitialStreamSemRequiresInitial	Checks that schema objects required by streaming semantics exist. This rule is triggered if: • An initial streaming semantic is used on a fact which is not initial in this job. You can either change the fact to initial and streaming, or select a different semantic. • A non-initial streaming semantic is used on a fact which is initial in this job. You can either change the fact to non-initial and streaming, or select a different semantic.
MissingMartTable	Checks that all expected actual and intmap tables exist in the mart. If a table, view or column is missing, Generate Schema before running the job.
NoMirrorMomBuildStateAlterSemantic	MomBuilder cannot be run in NoMirror mode if there is a state-altering semantic preceding it.
NoMultipleUndoFactSemantic	Checks that no more than one fact-combining semantic is run for each fact per job (between commit steps).
NonStateAlterNoStreamSem	Streaming extractions cannot have non-state-altering semantics.
NonTransFactHasTransIncrStreamingSemantic	Ensure that facts with Transactional Incremental Streaming semantics are transactional facts.
ObtainCampaignLockMatchReleaseCampaignLock	Tests whether each Obtain Campaign Lock job step has a matching Release Campaign Lock job step later in the job. No other backfeed lock step is between the two. Every step between an Obtain Backfeed Lock and a Release Backfeed Lock step must have On Error set to Abort or StartOver
ScrutinyError	Scrutiny is run during job validation. Refer to the error message for more information.
SeedFactNoRelFact	If a seed fact is involved in an explicit relation with a seed dimension for which a semantic is run, the semantic must also be applied to the fact. This error is triggered if the fact does not have the expect-

Exception	Test Condition
	ed semantic applied. The rule is checked at the commit step.
SeedFactSemanticRequiresSeedDimSemantic	If a seed fact has a semantic run against it, you must also run a semantic against its seed dimension.
SemanticBeforeCommit	If a dimension has extraction performed against it and any of the columns are integer mapped, a semantic must be performed against the dimension before the next commit.
SemanticRequired	If you have a streaming extraction step in the job, you must apply a streaming semantic before the commit step.
ShouldRunMombuilder	Mombuilder is required before a Commit step in a job if one of the following is true: • A semantic is run against a list-producing dimension • A semantic is run against a fact which is the fact for an explicit relation • A semantic is run against a seed fact
SQLStepForIntmapDimTest	Checks that there is a SQL step before the semantic, for dimensions with integer mapped columns. This rule checks for the last point under "Job Consistency" on page 315.
StreamingNoTruncation	Streaming extractions must be truncated and extracted before the semantic is applied.
StreamingSemanticOnly	If a streaming extraction step is used, there must be an associated streaming semantic on the fact or dimension. Also, you cannot run a streaming semantic against a non-streaming fact or dimension.
UncommittedFilterNotInJob	If there are any uncommitted filters in metadata, then they must be committed in this job, or the job must have no commit steps, or the job must include a Cancel step. This condition occur if filters exist from a previous incomplete or uncommitted job, or if an extraction step with filters follows a Commit step in the job definition.
UndoFactNoFollowSemantic	If a semantic is fact-combining - such as Undo Fact - it must follow another state-altering semantic. A fact-combining semantic cannot run against a fact

Exception	Test Condition
	before any other state-altering semantics are run against that fact.

Warnings

Table 128: Job Validation Warnings

Exception	Test Condition
FusionPreserveDimFactConfigError	If a semantic is being run on a fact that joins to fusion-enabled dimensions, and you are using Latest Dimension Value, Preserve Fusion (or no semantic at all) on all of those dimensions, then one of the following has to be true: • The fact is in an "Equal" state (indicating that, presumably, two fusion semantics have been run) • The fact semantic is fusion-enabled • The fact semantic is being run in force rebuild mode When this rule is triggered, it is possible to get the associated fact data into an invalid state. The following options are available to clear this warning and assist correct data loading: • Set the semantic run against the affected fact to Force rebuild of target table in the job. • Set the value of Fact Fusion/Fission to 1 in the semantic against the fact in the job. • Get the affected fact into an "Equal" state. This can be accomplished by running two consecutive jobs with non-initial semantics against the fact. This warning may be ignored if the job conforms to all the specifications for fusion-enabled jobs.
NoCommitStepsInJob	Checks whether there is at least one Commit or CommitBackfeed step in a job.
TruncAfterExtraction	A truncation step between an extraction and a state-altering semantic may lead to an unintentional condition in which data is not brought into the mart.