Developing Analytical Applications with Telco Analytics Manager

V4.0
Document ID: TAGN-06-4.0-01
Date Published: 9.10.03

Preface

In This Section

Using this Manual

Welcome to Developing Analytical Applications.

This manual covers building Analytical Applications with Telco Analytics Manager.

Before You Get Started

You should be familiar with the following:

- Your application architecture
- The CBU
- Designing or working with databases and data warehouses
- SQL

Who Should Read this Manual

This manual is for developers and project managers who are responsible for developing Analytical applications.

However, there are other topics covered in this manual that may interest other members of the project development team.

- Administrators

  You will find information about the different components that are involved in loading the CBU. You will also find information about how the analytic application works with the CID and Account Management applications. There is also information about customization you need to be aware of when dealing with deployed applications.
- Developers

  This manual is for developing Analytical applications. You learn how to customize the CBU. Building and using organization views is also covered in this manual. You also learn how to customize and integrate your Telco Analytics Manager with Account Management applications.
- Project Architect

  You can use the information in this manual to learn about how the CBU is customized and how you can modify the CBU and its components in order to meet the needs of your Telco Analytics Manager and Telco Service Manager. You might also find the information about organization views of interest.
- Project Manager

  You will find information about CBU customization. But what should interest you the most is the sections dealing with loading the CBU and working with organization views. You should also read the information about integrating TSM.

How this Manual is Organized

This manual contains the following chapters:

- **Overview of Developing Analytical Applications**

  This chapter covers the basics of the developing Analytical applications.

  It contains an overview of:

  - Customizing the Database
  - Synchronizing Data
  - Working with Organization Views
  - Loading Customer Dimensions
  - Building Simple Reports

- **Customizing the Database**

  This chapter covers customizing the CBU.

  It contains information about:

  - Adding Dimensions
  - Adding new Attributes

- **Customizing the Synchronization of Customer Data**

  This chapter covers developing the way the customer data is loading into the CBU.

  It contains information about:

  - Customizing the Notification Logic
  - Customizing the CID to CBU mapping

- **Working with Organization Views**

  This chapter covers working with organization views.

  It contains information about:

  - Creating Organization Views
  - Modifying Organization Views
  - Downloading and Uploading Organization Views

- **Using the CustDim Loader**

  This chapter covers using the CustDim Loader to load customer dimension data.

It contains information about:

- The CustDim Loader and its components
- Working with invoice files used to load the CBU
- Customizing invoice mapping

- **Creating Report Queries in SQL**

  This chapter covers using SQL to query the CBU.

  It contains information about:

  - Writing simple reports
  - Writing advanced reports

- **Deploying with Account Management Features**

  This chapter covers deploying Telco Analytics Manager along with an Account Management application.

  It contains information about:

  - How to integrate the Telco Analytics Manager with Account Management applications

What Typographical Changes and Symbols Mean

This manual uses the following conventions:

TYPEFACE	MEANING	EXAMPLE
Italics	Manuals, topics or other important items	Refer to *Developing Connectors*.
Small Capitals	Software and Component names	Your application uses a database called the CID.
Fixed Width	File names, commands, paths, and on screen commands	Go to `//home/my file`

Finding the Information You Need

edocs Telco Solutions comes with comprehensive documentation that covers all aspects of building TAM solutions. You should always read the release bulletin for late-breaking information.

Getting Started

If you are new to edocs Telco Solutions, you should start by reading *Introducing Telco Analytics Manager*. This manual contains an overview of the various components along with a list of the available features. It introduces various concepts and components you must be familiar with before moving on to more specific documentation. Once you have finished, you can read the manual that covers different aspects of working with the application. At the beginning of each manual, you will find an introductory chapter that covers concepts and tasks.

Designing Your Solution

While reading *Introducing Telco Analytics Manager*, you can begin to envision how the different components can address your solution's needs.

You can refer to *Developing Analytical Applications* for information about customizing the database, synchronizing data with Telco Service Manager (TSM), loading data from external invoice files, and other design issues. The *CBU Reference Guide* also gives you the information about how the information in your solution is managed and stored. You should also read the section on integrating Telco Analytics Manager with Telco Service Manager in *Developing Analytical Applications*.

Installing Your Analytical Application

You should start by reading the Release Bulletin. For detailed installation and configuring information, refer to *Installing Telco Analytics Manager*. This manual covers installing *Telco Analytics Manager* on one or more computers. It also contains the information you need to configure the different components you install.

Building Analytical Solutions

If you are designing and programming analytical applications, you have several different sources of information. If you are programming the user interface of the solution, you should read *Building Reports*. You can also refer to the *QRA API Specification* and the *QRA Configuration File Reference Documentation* for detailed information about the different components you can use to build reports which serve as the user interface. For configuring the various components, refer to *Installing Telco Analytics Manager* and sections in other documents that specifically deal with the component to be configured.

If you are working on the data warehouse side of TAM and are interested in how the information will be loaded into the data warehouse, you should read *Developing Analytical Applications*. For more information about the design and structure of the CBU, you should refer to the *CBU Reference Guide* along with the *CBU Reference* documentation for your database.

Integrating TAM and TSM

If you are involved in configuring your solution to work with Telco Service Manager, you should read *Introducing Telco Analytics Manager* for an overview of the components and how they interact. You should then read *Developing Analytical Applications* for information about synchronizing data between the Telco Analytics Manager and Telco Service Manager. In this manual, you will also find information about loading data in both the CBU and the Telco Service Manager.

Managing Telco Analytics Manager

If you are responsible for managing Telco Analytics Manager, you should read the *Installing Telco Analytics Manager* for information about configuring various components. *Administrating Telco Analytics Manager* covers what you need to know about managing your solution at runtime.

If You Need Help

Technical support is available to customers who have valid maintenance and support contracts with edocs. Technical support engineers can help you install, configure, and maintain your edocs application.

To reach the U.S. Service Center, located in Natick, MA (Monday through Friday 8:00am to 8:00pm EST):

- Telephone: 508.652.8400
- Toll Free: 877.336.3362
- E-support: support.edocs.com (This requires a one-time online registration)
- E-mail: support@edocs.com

When you report a problem, please be prepared to provide us the following information:

- What is your name and role in your organization?
- What is your company's name?
- What is your phone number and best times to call you?
- What is your e-mail address?
- In which edocs product did a problem occur?
- What is your Operating System version?
- What were you doing when the problem occurred?
- How did the system respond to the error?
- If the system generated a screen message, please send us that screen message.
- If the system wrote information to a log file, please send us that log file.

If the system crashed or hung, please tell us.

Contents

CHAPTER 1

Overview of Developing Analytical Applications

In This Section

About Developing Applications

You use Telco Analytics Manager to build your solution for your customer's needs in viewing and analyzing accounts with communication service providers.

Developing Analytical Applications involves:

- Customizing the database
- Synchronizing data
- Working with Organization Views
- Building Reports

Customizing the Database

The CBU contains customers billing and usage information.

Users can analyze their invoices using:

- Reference dimensions such as date/time, tariff, and service
- Customer-specific information such as organization views, contracts, and billing accounts

In order for your solution to meet your user's needs, you may need to customize the structure of the CBU.

Customizing the database involves:

- Adding new reference dimensions
- Localizing
- Managing charge type heirarchies
- Adding new attributes to tables

Synchronizing Data

Depending on how your Telco Analytics Manager is integrated and the deployed features, the CBU must be regularly synchronized with the CID to ensure that the data is coherent.

This synchronization is achieved in a very similar manner to the synchronization with back-office systems – using an asynchronous request process called the CID2CBU loader. Notifications are created whenever a change is made to the CID that must be synchronized with the CBU.

Customizing the synchronization of data involves:

- Customizing the notification logic
- Customizing the CID2CBU object mapping

Working with Organization Views

You can use organization views to create your own hierarchies. Not being dependent on the organization of information, organization views let you organize organizations and associated contracts the way you want them to be, not as dictated by your OSS.

Working with Organization Views involves:

- Creating organization views
- Modifying organization views
- Managing organization views

Loading Customer Dimension Data

When the only available customer data are in invoices, both the CID and the CBU must be populated with them.

This loading That loading is achieved by a batch processing of invoice files built on standard ISF components. The tool you use to do this is the CustDim Loader. Loading customer dime dimension data involves:

- Configuring the CustDim loader to process your invoice files
- Configuring the CustDim loader mapping to update the CID and CBU

Building Reports

The CBU is an open database. You can use any reporting tool you like to access information in the CBU for your users, even simple SQL queries.

Building reports involves:

- Writing simple SQL to query the CBU
- Writing advanced SQL to query the CBU

You also use the Query, Reporting, and Analysis Engine (QRA) to build reports. For more information about the QRA, refer to *Developing Reports*.

CHAPTER 2

Customizing the Database

In This Section

Adding New Reference Dimensions

You may need to add new reference axes of analysis on a fact table. This means adding a new reference dimension table. To add a new reference dimension table, you must respect the same pattern as the existing core tables.

1. Create the new dimension table with default dimension table attributes.

   Use a name like `CUSTOMDIM_DIM` for your new dimension table. The NMY prefix is reserved for core CBU tables.

2. Create required indexes and constraints of the reference dimension table.
3. Alter the FACT table to create a foreign key on the new dimension reference table

The CBU demo kit provides a sample of how to add a reference dimension table.

Localizing Reference Dimension Table Names

To localize the names of a reference dimension table and description

1 Create a new table in the data reference dimension tablespace (`NMY_CBU_REF_DATA`) containing the following:

- A foreign key pointing to the dimension table to localize
- A language code
- The localized name

2 Create the following in the index reference dimension tablespace (`NMY_CBU_REF_INDX`):

- Constraints:

  A primary key based on the foreign key and the language code
- Index:

  One index based on the language code and the localized name

Managing Charge Type Hierarchies

By default, the CBU does not manage charge types hierarchy.

If required, you can implement one of the following solutions depending on your needs:

- **Static hierarchy**

  A new dimension table is added to categorize the charge type.

  If more hierarchy levels are required, Integrator can either categorized the new dimension table or add new degenerated dimension on it.

  Advantages of this solution:

  - Easier reporting
  - Clear separation between charge type and category

  Disadvantages of this solution:

  - Hierarchy depth is constant and known
  - Category is required even for a flat charge type
  - Difficulty to provide reports with charge types of a different depth

    (all charge types should be designed with the same depth, even if it requires to implement fake category)
  - Does not allow charge values at different level of the same charge type category.

  For more information on a categorized charge type dimension table, refer to *Categorizing Values in Reference Dimension Tables* in this manual.

- **Dynamic hierarchy**

  Dynamic Hierarchy is implemented between charge types

  Advantages of this solution:

  - Allow reports on non constant hierarchy
  - Allow charge values at different levels of the same charge type hierarchy

  Disadvantages of this solution:

  - Reporting with Master/Detail is far more complex.
  - Query engine are difficult to use.
  - It is not possible to enforce pure category node without data. Types and categories are mixed in the same table.

  When using this design, we strongly recommend storing only addable values at different levels of the same charge type hierarchy.

  It is possible to store intermediate sums but you cannot mix both modes.

For more information about implementing hierarchies between charge types, refer to *Implementing Hierarchy Between Reference Dimension Table Values* in this manual

Categorizing Values in Reference Dimension Tables

You can to implement reference dimension hierarchy tables to categorize them (snowflake pattern).

To categorize a reference dimension table:

1. Create a new reference dimension table to store categories
2. Alter the initial dimension reference table to create a new foreign key pointing to the dimension table storing categories.

The CBU demo kit provides a sample of how to categorize a reference dimension table.

Adding New Degenerated Dimensions to FACT Tables

You can add a new axis of analysis on a fact table with unpredictable values that keep you from creating a new reference dimension table or merge this axis with an existing dimension (like destination number).

To add a new degenerated dimension:

1. Alter the fact table to create a new column to store the value of the new axis
2. Create an index on this column to analyze it

The index must be created on the `NMY_CBU_USAGE_INDX` or `NMY_CBU_INVC_INDX` tablespace depending on the fact table.

Implementing Hierarchy Between Reference Dimension Table Values

You can build a hierarchy between values of a reference dimension table.

To implement the hierarchy, you must:

1 Create a new hierarchy link table to store hierarchy links.

The CBU demo kit provides a sample of how to implement a hierarchy between reference dimension table values.

Adding New Attributes to FACT Tables

You can add a new attribute on a fact table for display or calculation purpose (for example `volume2`).

To add a new attribute on a fact table, alter the table and create the new column

Adding New Attributes to Reference Dimension Tables

You can add a new attribute on reference dimension table to store more detail information in it (for example, short description).

To add a new attribute to a reference dimension table, alter the table and create the new column.

Adding New Attributes to Customer Dimension Tables

You can add new attributes to actor, contract or billing account objects.

Supported types are:

- String
- Numeric
- Date

To add a new attribute to a customer dimension table, you must:

1 Alter the customer dimension table to add the new column

2 Customize the CID2CBU loader mapping policy to synchronize the attribute.

For more information about customizing the CID2CBU mapping, refer to *Customizing the BLM to CBU Mapping* in this manual.

Optimizing

If controls are implemented in your loading process, you can disable primary keys, alternate keys and foreign keys on fact tables.

If you do not use a reference dimension table, you can also disable the index related to this dimension on the fact table column.

Unsupported Customization

ACTION	COMMENT
Adding customer dimension table	Access to fact tables must always be done via contracts or billing accounts
Removing tables	All core tables provided in CBU must be kept even if not used
Removing or modifying core columns on any table	
Disabling constraints or indexes on customer dimension tables	
Removing core constraints or indexes on any table	You can disable a constraint or an index in fact table but never remove it

CHAPTER 3

Customizing the Synchronization of Customer Data

In This Section

Filtering Customers to Synchronize

About Customizing Notifications

Customizing notifications involves:

- Writing and deploying a new notification logic class
- Modifying the mapping between the CID and CBU

Customizing Notification Logic

You can customize this logic to meet your application's needs.

These notifications are used to synchronize data in the CBU. For instance, your CBU may only need information about a certain type of customer organization or a customer organization that has a specific type of contract. You want your notification logic to generate notifications only for changes to these types of organizations.

The core notification logic is in the following BLM external class:

```
com.netonomy.blm.external.NotificationLogic
```

Implementing your own notification logic involves:

- Writing a new class to implement your notification logic
- Deploying the class and declaring it in the BLM

Creating a New Notification Class

Writing a new class implementing your own notification logic involves:

- Defining a new package for your custom class
- Creating the new class extending the core class
- Writing your notification logic in the `isNotified` method to evaluate if notification is generated depending on organization attributes (including additional information)
- Compiling your class

Once you have written and compiled your class, you deploy it and declare it in the BLM

When creating your java class, we suggest declaring a Java Package to implement your class.

For example, `com.<yourclasspackage>.netonomy.blm.external` where `<yourclasspackage>` is the name of your company or the name of your customer.

To create a new class extending the core class

Here is an example of your new class:

```
package com.<yourclasspackage>.netonomy.blm.external;


import com.netonomy.blm.api.Organization.OrganizationF;

import com.netonomy.blm.external.NotificationLogic;

import com.netonomy.blm.interfaces.organization.OrganizationTypeCategoryIF;

import com.netonomy.blm.interfaces.organization.OrganizationTypeIF;


public class CustomNotificationLogic extends NotificationLogic

{


}
```

To code your notification logic

The notification logic is implemented in the `isNotified` method.

The `isNotified` method has the following parameter:

- `organization`: organization to test

The `isNotified` method returns a boolean specifying if the organization generates a notification.

The notification logic class is instantiated only once for the process and the different threads access the same instance. Your code must be thread safe.

Here is an example of the `isNotified` method in your class with the core notification logic code:

```
package com.<yourclasspackage>.netonomy.blm.external;
import com.netonomy.blm.api.Organization.OrganizationF;
import com.netonomy.blm.external.NotificationLogic;
import com.netonomy.blm.interfaces.organization.OrganizationTypeCategoryIF;
import com.netonomy.blm.interfaces.organization.OrganizationTypeIF;
public class public class CustomNotificationLogic extends NotificationLogic
{

    public boolean isNotified(OrganizationF org)
    {
 // Replace the following code by your own
 OrganizationTypeIF type = org.getType(); // mandatory -> should not be null
        OrganizationTypeCategoryIF  cat  = type.getCategory();
        if (cat!=null)
        {
            String cat_code = org.getType().getCategory().getCode();
            if (cat_code!=null)
            {
                if (cat_code.equals("FLAT_CUSTOMER") || cat_code.equals("LARGE_CUSTOMER"))
                {
                    return true;
                }
            }
        }
        return false;
    }
}
```

To compile your class

To compile your class, you need to make sure the following `jar` files are in your classpath:

- `lib/nmycore.jar`
- `lib/nmyutil.jar`

Integrating the Notification Class

To deploy your class

1. Create sub folders consistent with your package name. For example create a folder called `classes/com/<yourclasspackage>/netonomy/blm/external`.
2. Copy your compiled class to this folder.

To declare your class

1. Go to `<home_dir>/classes/nmycfg/blm`.
2. Open the `external_custom.xml` customization file.
3. Find the `class` element with `default` attribute equal to `com.netonomy.blm.external.NotificationLogic`.
4. Enter the name of your custom class as the value of the `custom` attribute.

   Example:

   ```
   <class default=" com.netonomy.blm.external.NotificationLogic" custom="
   com.<yourclasspackage>.netonomy.blm.external.CustomNotificationLogic"/>
   ```
5. Save your changes.

Customizing the BLM to CBU Mapping

The CID2CBU loader synchronizes CBU customer dimensions with the CID tables. The loader sets CBU dimensions with BLM object attributes.

The mapping policy is declared in the CID2CBU `mapping.xml` configuration file This file is located in `<home_dir>/config/cid2cbu`.

This file specifies for every objects:

- The complete name of the class implementing the mapping interface
- For every custom columns, the name of its mapping method in the class.

The file syntax is as follows:

```
<mapping>
  <object name=”object Name” table=“CBU Table name storing the object” impl = “mapping class name”>
    <attr column=“CBU Table custom column name” acc= “custom mapping method name”/>
  </object>
</mapping>
```

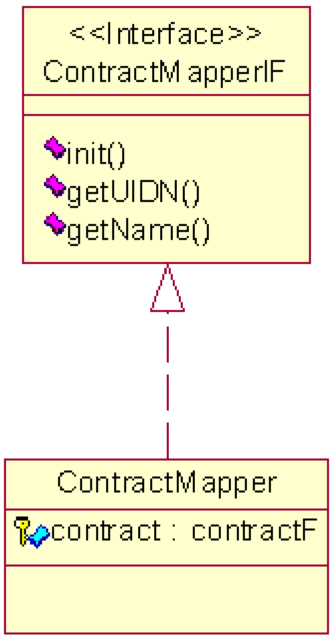
You can customize the mapping of the contents of the CBU and CID to:

- Modify the default mapping of CBU dimensions core attributes.
- Specify the mapping of CBU dimensions custom attributes

Default Mapping

For every object mapped in the CBU, an external class implements a standardized interface to specify the default mapping of core attributes (Name and UIDN for most objects, LOGIN for the user object, PATH for orgview objects)

The following class diagram presents an example of the interface and of the default implementation of contract mapping.



Refer to the *CID2CBU Object Mapping Reference Documentation* for the detailed description of all interfaces and default implementation classes (including default mapping specification).

Modifying the Default Mapping

You may have to modify the default mapping of a core attribute. For instance, you may need to modify the source of information or to modify the format in the CBU.

For example, the CBU Contract Name is set to the exact contract line number in the CID by default. For readability, you may want to add blanks between the numbers.

Modifying default mapping of a core attribute involves:

1. Writing a new mapping class to extend the existing class and implement the related interface
2. Overwriting in this class the mapping method of the attribute
3. Declaring your class in CID2CBU `mapping.xml` configuration file
4. Deploying the class

Specifying the Mapping of a Custom Attribute

Your solution may require the declaration of new columns in customer dimension tables.

Specifying the mapping of this new attribute involves:

- Writing a new mapping class to extend the existing class and implement the related interface
- Implementing the new method to map the custom attribute
- Declaring your class and your custom attribute mapping in the CID2CBU `mapping.xml` configuration file
- Deploying the class

The following sample code specifies mapping of a new `RATEPLAN` column of the `NMY_CONTRACT` table with the current contract rate plan in the CID.

```
package com.<yourclasspackage>.netonomy.cbu.cid2cbu.external;

import com.netonomy.blm.api.Contract.ContractF;

import com.netonomy.blm.interfaces.rateplan.RatePlanIF;

import com.netonomy.cbu.cid2cbu.external.ContractMapper;


public class CustomContractMapper extends ContractMapper

{

 // The return parameter type must be consistent with the column type in CBU

    public String getRatePlanCode() {

return contract.getMyRatePlan().getCode();

    }

}
```

When creating your java class, we suggest declaring a Java Package to implement your class.For example, `com.<yourclasspackage>.netonomy.cbu.cid2cbu.external` where `<yourclasspackage>` is the name of your company or the name of your customer.

To compile your class, you make sure the following jar files are in your classpath:

- `lib/nmycore.jar`
- `lib/nmyutil.jar`
- `lib/nmycid2cbu.jar`

To deploy your class, you must create a sub folder consistent with your package name (example: `classes/com/<yourclasspackage>/netonomy/cbu/cid2cbu/external`) and copy your compiled class to this folder.

If you need to totally modify the default mapping of an object, you can directly write a new class implementing the interface instead of extending the core implementation

Example of Mapping Declarations

The following `mapping.xml` sample configuration file shows the declaration of both the previous class and the custom attribute.

```
<mapping>

  <object name="contract" table="NMY_CONTRACT_DIM" impl =
"com.<yourclasspackage>.netonomy.cbu.cid2cbu.external.CustomContractMapper">

  <attr column="CONTRACT_RATEPLAN" acc= "getRateplanCode"/>

</object>

…….

</mapping>
```

CHAPTER 4

Working with the Report Manager

In This Section

About the Report Manager

You use the Report Manager to allow your users to save their reports. Not only can they save reports, users can see a list of saved reports, rename them and delete them when they are no longer needed.

The Report Manager is based on the Web File System (WFS) component. This component allows you to use the application server to save and manage report files.

Working with the Report Manager involves:

- Saving Reports
- Listing Saved Reports
- Renaming Reports
- Deleting Reports

For information about configuring and administrating the Report Manager, refer to *Administrating Telco Analytics Manager*.

Saving Reports

Saving a report involves:

1. Getting the report manager to use
2. Getting the report to save
3. Creating the file
4. Validating the file
5. Getting any errors that occurred for further processing

The `logic_saveReport` code in the `logic_report.jsp` shows how to save a report:

Getting the report manager	`{`
	`PermanentFileManager fileMgr = getCurrentPermanentFileManager (session, request, jspHelper);`
	`String reportName = (String)request.getParameter("report_name");`

Get the report to save	`Report toSave = getCurrentReport (session, request, jspHelper, getCurrentCacheFileManager (session, request, jspHelper));`
	`jspHelper.doNotSend ("report_name");`

Create the file, get the data, then close the file	`FileDescriptorIF saveTo = fileMgr.createFile(reportName);`
	`BufferedOutputStream toWrite = new BufferedOutputStream (saveTo.getOutputStream());`
	`try`
	`{`
	`toSave.save (toWrite);`
	`} finally`
	`{`
	`toWrite.close();`
	`}`

Validate the content	`fileMgr.commitFile(saveTo);,`

Get the errors	`request.getRequestDispatcher(jspHelper.getUrlAndMoveToPage ("ON_OK")).forward(request, response);`
	`} catch (FileManagerException except)`
	`{`
	`request.setAttribute("Exception",except);`
	`request.getRequestDispatcher(jspHelper.getUrlAndMoveToPage ("ON_EXCEPTION")).forward(request, response);`
	`}`
	`}`

Listing Reports

Listing the saved reports involves:

1 Getting the report manager to use

2 Getting the list of reports

3 Getting any errors that occurred for further processing

The `logic_getSavedReports` code in the `logic_report.jsp` shows how to delete a report:

Getting the report manager	`{`
	
	`PermanentFileManager fileMgr = getCurrentPermanentFileManager (session, request, jspHelper);`

Get the reports to display	`String[] names = fileMgr.listFileNames();`

Get the errors	`if( names!=null)`
	
	`request.setAttribute("report_list", names);`
	
	`request.getRequestDispatcher(jspHelper.getUrlAndMoveToPage ("ON_OK")).forward(request, response);`
	
	`} catch (FileManagerException except)`
	
	`{`
	
	`request.setAttribute("Exception",except);`
	
	`request.getRequestDispatcher(jspHelper.getUrlAndMoveToPage ("ON_EXCEPTION")).forward(request, response);`
	
	`}`

Renaming Reports

Renaming a saved report involves:

1. Getting the report manager to use
2. Getting the old and new names
3. Saving the report with the new name
4. Getting any errors that occurred for further processing

The `logic_renameReport` code in the `logic_report.jsp` shows how to delete a report:

Getting the report manager	`{`
	
	`PermanentFileManager fileMgr = getCurrentPermanentFileManager (session, request, jspHelper);`

Get the old and new name	`String reportName = (String)request.getParameter("report_name");`
	
	`String oldReportName = (String)request.getParameter("old_report_name");`
	
	`jspHelper.doNotSend ("report_name");`
	
	`jspHelper.doNotSend ("old_report_name");`

Rename the report	`fileMgr.renameFile (oldReportName, reportName);`

Get the errors	`request.getRequestDispatcher(jspHelper.getUrlAndMoveToPage ("ON_OK")).forward(request, response);`
	
	`} catch (FileManagerException except)`
	
	`{`
	
	`request.setAttribute("Exception",except);`
	
	`request.getRequestDispatcher(jspHelper.getUrlAndMoveToPage ("ON_EXCEPTION")).forward(request, response);`
	
	`}`

Deleting Reports

Deleting a saved report involves:

1 Getting the report manager to use

2 Getting the report to remove

3 Removing the report

4 Getting any errors that occurred for further processing

The `logic_removeReport` code in the `logic_report.jsp` shows how to delete a report:

Getting the report manager	`{`
	`PermanentFileManager fileMgr = getCurrentPermanentFileManager (session, request, jspHelper);`
	`String reportName = request.getParameter ("report_name");`
	`jspHelper.doNotSend ("report_name");`

Get the report to remove	`String reportName = request.getParameter ("report_name");`
	`jspHelper.doNotSend ("report_name");`

Remove the report	`if (fileMgr.removeFile(reportName))`

Get the errors	`{`
	`request.getRequestDispatcher(jspHelper.getUrlAndMoveToPage ("ON_OK")).forward(request, response);`
	`}`
	`else`
	`{`
	`request.getRequestDispatcher(jspHelper.getUrlAndMoveToPage ("ON_ERROR")).forward(request, response);`
	`}`
	`} catch (FileManagerException except)`
	`{`
	`request.setAttribute("Exception",except);`
	`request.getRequestDispatcher(jspHelper.getUrlAndMoveToPage ("ON_EXCEPTION")).forward(request, response);`
	`}`

CHAPTER 5

Working with Organization Views

You can use organization views to create your own hierarchies. This gives you the freedom to organize your information as you see fit, not as it is stored in your system.

For instance, your OSS may store information based on profit centers or departments. When your data is stored like this, you may have a hard time managing information based upon employee location, such as an office or city. Let's say your OSS systems keep track of your contracts and users based on departments (sales, presales, accounting, product development, and so on). You may find it difficult to manage this type of information when you want to visualize and manage users and costs based on branch office locations, where employees from several different departments work together.

You can use organization views to create your own hierarchies. Not being dependent on the organization of information, organization views let you organize organizations and associated contracts the way you want them to be, not as dictated by your OSS.

An organization view has the following components:

- A unique user reference
- A name
- Levels and sublevels
- Contracts

Working with Organization Views involves:

- Creating organization views
- Changing owners
- Deleting organization views
- Adding levels
- Moving levels
- Removing levels
- Assigning contracts to levels
- Removing contracts from levels
- Downloading organization views
- Uploading organization views

The MyWeb channel JSPs use the organization view feature. This channel uses organization views called cost structures. This is an organization view created specifically to group users and contracts for cost analysis, no matter how the OSS organizes the data. All of the examples in this section show you how to use the organization view to create cost structure views.

About Working with Organization Views

You can use organization views to create your own hierarchies. This gives you the freedom to organize your information as you see fit, not as it is stored in your system.

For instance, your OSS may store information based on profit centers or departments. When your data is stored like this, you may have a hard time managing information based upon employee location, such as an office or city. Let's say your OSS systems keep track of your contracts and users based on departments (sales, presales, accounting, product development, and so on). You may find it difficult to manage this type of information when you want to visualize and manage users and costs based on branch office locations, where employees from several different departments work together.

An organization view has the following components:

- A unique user reference
- A name
- Levels and sublevels
- Contracts

Working with Organization Views involves:

- Creating organization views
- Changing owners
- Deleting organization views
- Adding levels
- Moving levels
- Removing levels
- Assigning contracts to levels
- Removing contracts from levels
- Downloading organization views
- Uploading organization views

The MyWeb channel JSPs use the organization view feature. This channel uses organization views called cost structures. This is an organization view created specifically to group users and contracts for cost analysis, no matter how the OSS organizes the data. All of the examples in this section show you how to use the organization view to create cost structure views.

Creating Organization Views

Creating organization views involves:

1 Getting the name and user reference from the user.

2 Getting the additional parameters.

3 Creating the organization view.

The sample JSP code shows how to create a cost structure organization view:

Get the name and user reference	`String orgViewName = request.getParameter("name");` `String orgViewUserReference = request.getParameter("refnumber");`
Fill the additional parameters and catch any errors	`ParameterIF[] values = new ParameterIF[0];` `ParameterIF[] parameters = ObjectRefMgr.getOptionalParameterDescriptors (((ObjectId)DescriptorOidIF.ACTION_DO_ADD_ORGVIEW), values);` `try` `{` `fillParametersWithRequest (parameters, request);` `}` `catch (BadValueException ex)` `{` `response.sendRedirect(jspHelper.encodeURLFunct ("GLOBAL.PARAMETER_ERROR", null, true));` `jspHelper.exitJSP();` `}`
Create the cost structure	`SessionF  blmSession = jspHelper.getBlmSession();` `UserF user = blmSession.getUserF();` `LevelF level = user.getLevel();` `level.doAddOrgView(orgViewUserReference, orgViewName, parameters);`

Changing Organization View Owners

Creating organization views involves:

1 Getting the user and owned organization views.

2 Getting the new owner.

3 Changing the ownership of the organization view.

The sample JSP code shows how to change the ownership of an organization view:

Get the user, the user's organization views, and the other users of the current user's organization level.	`//get the user of the session`
	`UserF  user  = blmSession.getUserF();`
	`//get the orgviews of the user`
	`OrgViewIF[] orgView = user.getOrgViews();`
	`//get the members at level of user`
	`UserF[] users = user.getLevel().getUsers();`

Create a new owner of the organization view	`//get the ObjectId of new user`
	`ObjectId object = ObjectId.instantiate(member);`
	`//create a new user`
	`UserF newOwner = new UserF(blmSession,object);`

Change the ownership of the organization view to the new user	`//change owner of orgView`
	`orgView[0].doChangeOwner(newOwner);`

Deleting Organization Views

Deleting organization views involves:

1. Getting the organization view to delete.
2. Testing the organization view to see if it can be deleted.
3. Deleting the organization view.

The sample JSP code shows how to delete the current organization view:

Get the current organization view	`OrgViewIF orgView = getCurrentOrgView(request, response, jspHelper);`
Test to see if the organization view can be removed	`if (orgView.isRemovable())`
Delete the organization view	`orgView.doRemove();`

Managing Levels

Working with organization views levels involves:

- Adding levels
- Moving levels
- Removing levels
- Assigning contracts to levels
- Removing contracts from levels

Adding Levels

Adding a level to an organization view involves:

1 Getting the name.

2 Getting the parent level.

3 Getting the additional parameters of the level to add.

4 Creating the level in the organization view.

The sample JSP code shows how to add a level to a cost structure organization view:

Get the name	`String orgViewName = request.getParameter("name");`

Get the parent level	`OrgViewIF   orgView;`
	`ObjectId orgViewId = ObjectId.instantiate (request.getParameter ("orgviewId"));`
	`orgView = (OrgViewIF)ObjectMgr.getOrgViewNode (orgViewId);`

Fill the additional parameters and catch any errors	`  // Get optional parameters list`
	`  ParameterIF[] values = new ParameterIF[0];`
	`  ParameterIF[] parameters = ObjectRefMgr.getOptionalParameterDescriptors (((ObjectId)DescriptorOidIF.ACTION_DO_ADD_ORGVIEWLEVEL), values);`
	`  try`
	`  {`
	`    fillParametersWithRequest (parameters, request);`
	`  }`
	`  catch (BadValueException ex)`
	`  {`
	`    response.sendRedirect(jspHelper.encodeURLFunct ("GLOBAL.PARAMETER_ERROR", null, true));`
	`    jspHelper.exitJSP();`
	`  }`

Create the level	`OrgViewLevelIF newLevel = orgViewNode.doAddOrgViewLevel(orgViewName, parameters);`

Moving Levels

Moving a level in the same organization view involves:

1 Getting the level to move

2 Getting the new parent level

3 Moving the level

The sample JSP code shows how to move a level in a cost structure organization view:

Get the level to move	`OrgViewNodeIF   orgViewLevel;`
	`ObjectId orgViewNodeId = ObjectId.instantiate (request.getParameter ("orgviewNodeId"));`
	`orgViewLevel = ObjectMgr.getOrgViewNode (orgViewNodeId);`

Get the destination parent level	`ObjectId parentId = ObjectId.instantiate(request.getParameter("orgviewnodeparentId"));`
	`OrgViewNodeIF orgViewNodeParent = ObjectMgr.getOrgViewNode(parentId);`

Call the BLM API to move the level	`orgViewLevel.doChangeParent(orgViewNodeParent);`

Removing Levels

Removing a level from an organization view involves:

1 Getting the level to remove

2 Calling the function to remove the level

The sample JSP code shows how to remove a level from a cost structure organization view:

Get the level	`OrgViewNodeIF   orgViewLevel;`
	`ObjectId orgViewNodeId = ObjectId.instantiate (request.getParameter ("orgviewNodeId"));`
	`orgViewLevel = ObjectMgr.getOrgViewNode (orgViewNodeId);`

Call the remove function	`orgViewLevel.doRemove();`

Assigning Contracts

Assigning contracts to an organization level involves:

1 Getting the level

2 Getting the contracts to assign
3 Assigning each contract

The sample JSP code shows how to assign a contract to a level of a cost structure organization view:

Get the level	`OrgViewNodeIF   orgViewLevel;`
	`ObjectId orgViewNodeId = ObjectId.instantiate (request.getParameter ("orgviewNodeId"));`
	`orgViewLevel = ObjectMgr.getOrgViewNode (orgViewNodeId);`

Get the contracts to assign	`ObjectId []contractIds = ObjectId.instantiate (request.getParameterValues ("contractId"));`
	`  for (int i=0;i<contractIds.length;i++)`
	`    {`
	`    curContract = new ContractF (blmSession, contractIds[i]);`

Assign each contract and catch any errors	`try`
	`      {`
	`      orgViewLevel.doAddContract (curContract);`
	`      }`
	`    catch (Exception e)`
	`      {`
	`       response.sendRedirect(jspHelper.encodeURLFunct ("GLOBAL.LOGIC_ERROR", null, true));`
	`      jspHelper.exitJSP();`
	`      }`
	`    }`

Removing Contracts

Removing contracts from an organization view involves:

1 Getting the level
2 Getting the contracts to remove
3 For each contract, call the `doRemove` function

The sample JSP code shows how to remove a contract from a level of a cost structure organization view:

Gets the level	`OrgViewNodeIF   orgViewLevel;`
	`ObjectId orgViewNodeId = ObjectId.instantiate (request.getParameter ("orgviewNodeId"));`
	`orgViewLevel = ObjectMgr.getOrgViewNode (orgViewNodeId);`

Gets the contracts to remove	`ObjectId []contractIds = ObjectId.instantiate (request.getParameterValues ("contractId"));`
	`  for (int i=0;i<contractIds.length;i++)`
	`    {`
	`    curContract = new ContractF (blmSession, contractIds[i]);`

Remove each contract from the level and catch any errors	`    try`
	`      {`
	`      orgViewLevel.doRemoveContract (curContract);`
	`      }`
	`    catch (Exception e)`
	`      {`
	`       response.sendRedirect(jspHelper.encodeURLFunct ("GLOBAL.LOGIC_ERROR", null, true));`
	`      jspHelper.exitJSP();`
	`      }`
	`    }`

Downloading and Uploading Organization Views

Telco Analytics Manager also lets you download and upload organization views.

The organization views are stored in XML files that you can easily save and edit. Once you are finished making your changes, you upload them into Telco Analytics Manager. Once finished uploading, you can begin using them right away.

You can modify this structure to suit your needs. Your JSPs can use the different BLM APIs to create complex organization views for your application.

About the Organization View XML File

By default, the organization view XML file contains the following elements:

- Organization view
- Reference
- Levels
- Contracts

The `<organization_view>` tag contains information on the organization view. The syntax is:

```
<organization_view name="name">
```

<ORGANIZATION_VIEW> ELEMENT	
Attributes: "name"	The name of the organization view.
Contents	One <reference> element One or more <level> elements

The `<reference>` tag contains the unique user-determined name for the organization view. The syntax is:

```
<reference>
```

<REFERENCE> ELEMENT	
Attributes: none	

The `<level>` tag contains the name of the organization view level. The syntax is:

```
<level name="name">
```

<LEVEL> ELEMENT	
Attributes: "name"	The name of the level.
Contents	One or more <level> elements One or more <contract> elements

The `<contract>` tag contains information on the organization view. The syntax is:

```
<contract>
```

<CONTRACT> ELEMENT	
Attributes: none	
Contents	The contract number

An example of the organization view XML file for cost structures:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<cost_structure name="Project One">
          <reference>View by sector</reference>
          <level name="Sector 1">
                    <contract>0660100026</contract>
                    <contract>0660100027</contract>
                    <level name="Sector 1.1">
                              <contract>0660100101</contract>
                    </level>
          </level>
          <level name="Sector 2">
                    <contract>0660100028</contract>
                    <contract>0660100102</contract>
          </level>
</cost_structure>
```

Overview of Using Organization View XML Files

One of the features of organization views is to easily download an existing organization view in XML.

Working with organization view XML files involves:

1 Downloading an organization file in XML

2 Editing the file

3 Uploading the XML into Telco Analytics Manager

Downloading Organization Views

Downloading a cost structure organization view involves:

1 Getting the cost structure to download

2 Writing the HTTP header for the browser to open the *Save As…* dialog box

3 Writing the XML header

4 Writing basic cost structure information

5 Browsing the cost structure hierarchy

6 For each level, writing the contract in XML format

The XML file is generated directly in the JSP (just like HTML pages), but unlike HTML, line breaks have a significant meaning in XML, especially at the beginning.

If XML editors do not recognize your generated file, make sure that the JSP does not have '\n' between Java code delimiters ('`<%`', '`%>`')

Get the cost structure	`<%`
	`SessionF  blmSession = jspHelper.getBlmSession();`
	`UserF user = blmSession.getUserF();`
	`OrgViewIF[] orgviews = user.getOrgViews();`
	`  // Gets the first cost structure of the list`
	`OrgViewIF orgView = orgviews[0];`

Write the HTTP header	`response.addHeader ("Content-Disposition", "attachment; filename=cost_structure_"+orgView.getUserReference()+".xml");`

Write the XML header	` // Notice the header is directly after %>, no new line.`
	`%><?xml version="1.0" encoding="<%=jspHelper.getEncoding()%>"?>`

Write the name and reference of the cost structure in XML	`<cost_structure name="<%=orgView.getName()%>">`
	`  <reference><%=orgView.getUserReference ()%></reference>`

Recursively write contract and sublevels	`<%displayContracts (out, "  ");%>`
	`<%displaySubLevels (out, "  ");%>`

Close the XML tag	` </cost_structure>`

As we must recurse through the cost structure hierarchy, the main code must be written in callable functions. The following functions are used and declared in this sample:

- `displayContracts`
- `displaySubLevels`

`displayContracts` Function

Write down the contract of the cost structure level	`<%!`
	`protected void  displayContracts (javax.servlet.jsp.JspWriter out, OrgViewNodeIF orgView, String pad) throws IOException`
	`{`

Get the contracts of the level	`ContractF[] contracts = orgView.getContracts ();`
For each contract,	`if (contracts!=null)`
	`{`
	`StringBuffer  buff=new StringBuffer();`
	`for (int i=0;i<contracts.length;i++)`
	`{`
Write the phone number between the contract tags	`buff.setLength(0);`
	`buff.append(pad).append("<contract>");`
	`buff.append(contracts[i].getPhoneNumber());`
	`buff.append ("</contract>");`
	`out.println(buff.toString());`
	`}`
	`}`
End function	`}`

displaySubLevels Function

Browse each level of the cost structure	`protected void  displaySubLevels (javax.servlet.jsp.JspWriter out, OrgViewNodeIF orgView, String pad) throws IOException`
	`{`
Get the sublevels of the level	`OrgViewLevelIF[] subLevels = orgView.getChildren ();`
	`if (subLevels!=null)`
	`{`
For each sublevel,	`// increase oas if 2 spaces so that sub-levels will be written shifted to the right in the file.`
	`String  newPad = pad+"  ";`
	`StringBuffer  buff=new StringBuffer();`
	`for (int i=0;i<subLevels.length;i++)`
	`{`
	`buff.setLength(0);`
Write the sub-level name tag.	`buff.append (pad).append("<level name=\"");`
	`buff.append (subLevels[i].getName()).append("\" >");`
	`out.println(buff.toString());`
Write the sublevel contracts	`displayContracts (out, subLevels[i], newPad);`

Then the sublevel sublevels	`    displaySubLevels (out, subLevels[i], newPad);`

Close the level tag	`    buff.setLength(0);`
	`    buff.append (pad).append("</level>");`
	`    out.println(buff.toString());`
	`    }`
	`  }`

End function	`}`

Uploading Organization Views

Uploading creates new organization views. If you modify an existing organization view then upload it, the new organization view replaces the existing one.

There is no history of organization views kept in the CID.

Uploading an organization view involves:

1 Writing an HTML form to upload the XML file

2 Writing a `logic_handler` JSP to do one of the following:

- Uploads the cost structure
- Updates an existing organization view

Writing the HTML Form

This form will be recognized by the browser and will allow the customer to choose and upload its file to the server.

The sample JSP code shows how to write the form.

Declare the form tag	`<!—Notice the mandatory enctype=”…” attribute`
	`<form name="upload_org_view" action="<%=jspHelper.encodeURLFunct("UPLOAD", null, false, true)%>" enctype="multipart/form-data" method="post">`
	`<table>`

Put the “File” button	`<tr>`
	`  <td class="listAText">`
	`    <input type="file" name="filename" >`
	`  </td>`
	`</tr>`

Put the submit button	`<tr>`
	`  <td class="listAText">`
	`    <input type="submit" value="<%=jspHelper.localize ("submit_txt")%>">`
	`  </td>`
	`</tr>`

Close the form	`</table>`
	`</form>`

Uploading a New Organization View

The file sent by the user must be correctly decoded and inserted into a new cost structure.

Uploading a new cost structure involves:

- Retrieving the request data
- Finding the beginning of the file in the request
- Parsing the XML using the standard Java XML parser
- Determining if it is update or new organization view
- Creating a transaction
- Creating the cost structure
- Browsing the XML and for each level:
  - Creating the sublevels
  - Checking that the defined contracts exist in the organization
  - Attaching them to the level
- Persisting the new cost structure.
- Submitting or rollbacking the transaction when an error occurs.

Load the request content into the buffer	`SessionF  blmSession = jspHelper.getBlmSession();`
	`UserF logged =blmSession.getUserF();`
	`OrganizationF  org = logged.getLevel().getHierarchyRoot().getOrganization();`
	
	`  StringWriter  buffer = new StringWriter ();`
	`  BufferedReader  reader = new BufferedReader (request.getReader());`
	`  String  readString=new String();`
	
	`  while (readString!=null)`
	`    {`
	`    readString = reader.readLine ();`
	`    if  (readString!=null)`
	`      buffer.write (readString);`
	`    }`

Search the XML file in the request	`// We now search the beginning of the xml in the string`
	`String req = buffer.toString ();`
	`int startXml = req.indexOf("<?xml");`
	`int endXml = req.lastIndexOf(">");`
	`if ((startXml!= -1) && (endXml!=-1))`
	`{`
	`req = req.substring(startXml, endXml+1);`
	`}`
Parse the xml	`DocumentBuilder parser = DocumentBuilderFactory.newInstance().newDocumentBuilder ();`
	`Document  xml;`
	`xml = parser.parse (new InputSource ( new StringReader (req)));`
Make sure the user reference is unique	`String userRef;`
	`Element elem = xml.getDocumentElement ();`
	`// Gets the first child tag named reference`
	`NodeList  list = elem.getElementsByTagName (“reference”);`
	`userRef =  list.item(0).getFirstChild().getNodeValue();`
	`// Check if the user reference already exists`
	`if (org.getOrgViewByReference(userRef)!=null)`
	`{`
	`// Yes it exists already => error`
	`Hashtable urlParameters = new Hashtable ();`
	`urlParameters.put ("ex_msg", jspHelper.localize ("known_user_reference", new Object[]{userRef}));`
	`response.sendRedirect (jspHelper.encodeURLFunct ("GLOBAL.LOGIC_ERROR", urlParameters, true));`
	`return;`
	`}`
Begin the transaction	`UserTransaction transaction= blmSession.getUserTransaction();`
	`transaction.begin();`
	`try   // For transaction support`
	`{`
Creates the cost structure	`// Get the name of the cost structure`
	`String name;`
	`Element elem = xml.getDocumentElement ();`
	`name = elem.getAttribute (“name”);`
	`OrgViewIF orgView=null;`
	`// Creates the organization_view`
	`orgView = logged.getLevel ().doAddOrgView (userRef, name, null);`

Fills the cost structure with the xml data. (Function shown below)	`Boolean filled = fillLevel (org, orgView, xml.getDocumentElement());`
	`if (filled==false)`
	`{`
	`// Some mistakes where made with the xml`
	`// Or some defined contracts do not exist in the organization`
	
	`// We rollback the whole transaction`
	`transaction.rollback();`
	`transaction=null;`
	
	`response.sendRedirect (jspHelper.encodeURLFunct ("GLOBAL.INTERNAL_ERROR", null, true));`
	`return;`
	`}`

Commit the transaction when no errors occured	`else`
	`{`
	`transaction.commit();`
	`transaction=null;`
	`request.getRequestDispatcher(jspHelper.getUrlAndMoveToPage ("ON_OK")).forward(request, response);`
	`}`

Or rollback it should any exception occurs	`} finally`
	`{`
	`if (transaction!=null)`
	`transaction.rollback();`
	`}`

As we need to recurse through the XML in order to create levels and sublevels, the java code has been put in the following function:

Function that creates a new level and associates contracts to it	`<%!`
	`protected boolean fillLevel  (OrganizationIF org, OrgViewNodeIF level, Element xml, String contractTxt, String levelTxt, String nameTxt)`
	`{`

Find all sub-levels and contracts of the current level.

Put them in distinct arrays

```
  // First adds the contract to the current level

  NodeList  nodelist = xml.getChildNodes ();

  ArrayList listPhones = new ArrayList ();

  ArrayList subLevels = new ArrayList ();


    // Fills the list of phones and sublevels

  if  ((nodelist!=null) && (nodelist.getLength()>0))

  {

    String phoneValue;

    for (int i=0;i<nodelist.getLength();i++)

    {

      if (nodelist.item (i) instanceof Element)

      {

      if (“contract”.equals(nodelist.item(i).getNodeName()))

        {

          // The child element is a contract declaration

        phoneValue=((Element)nodelist.item(i)).getFirstChild ().getNodeValue();

        listPhones.add (phoneValue);

        }

      else if (“level”.equals(nodelist.item(i).getNodeName()))

        {

          // It’s a level declaration

        subLevels.add (nodelist.item(i));

        }

      }

    }

  }
```

Prepare the check for contract existence by doing a generic search

```
  if (listPhones.size()>0)

    {

      //Search in the organization

      FilterIF filter =
ObjectRefMgr.getFilterByCode("CORE_INTORG_CONTRACTBYLINENUMBERS");

      ParameterIF []criteria = filter.getCriteria(FilterIF.ALL);

      ParameterIF nodeId =
ParameterHelper.getParameterByCode(criteria,"CORE_C_ORGID");

      ((ValueDynamicIF)nodeId).setDynamicValue(org.getIdentifier());

      ParameterIF subLevelsParam =
ParameterHelper.getParameterByCode(criteria,"CORE_C_SEARCHINSUBLEVELS");

      ((ValueSimpleIF)subLevelsParam).setBooleanValue(new Boolean(true));

      ParameterIF line =
ParameterHelper.getParameterByCode(criteria,"CORE_C_LINENUMBERS");
```

Check the user reference unicity

```
    String userRef;
    Element elem = xml.getDocumentElement ();
      // Gets the first child tag named reference
    NodeList  list = elem.getElementsByTagName (“reference”);
    userRef =  list.item(0).getFirstChild().getNodeValue();
        // Check if the user reference already exists
    if (org.getOrgViewByReference(userRef)!=null)
      {
        // Yes it exists already => error
      Hashtable urlParameters = new Hashtable ();
      urlParameters.put ("ex_msg", jspHelper.localize ("known_user_reference", new
Object[]{userRef}));
      response.sendRedirect (jspHelper.encodeURLFunct ("GLOBAL.LOGIC_ERROR",
urlParameters, true));
      return;
      }
```

Fills the list of contracts in the request criteria, then perform the search

```
          // Fill the parameters with the node value
      ValueListIF list = (ValueListIF)line;
      ParameterIF param;
      for (int i=0;i<listPhones.size();i++)
        {
        param = list.createParameterValue();
        param.setValue((String)listPhones.get(i));
        list.add (param);
        }
      list.setFilled(true);


      ContractF[] contracts = ContractF.search (filter);
```

Return an error if a contract is defined in the xml file but does not exist in the organization

```
        // Check all the contracts are in the organization
      if (contracts.length!=listPhones.size())
        {
          // Size mismatch => Error
        return false;
        }
```

Adds the contracts to the level otherwise

```
      // All the contracts are defined in the org.
    for (int i=0;i<contracts.length;i++)
      {
      level.doAddContract (contracts[i]);
      }
    }
```

Now browse all sub-levels…

```
      // Now recurse to all sub-levels
    if  ((subLevels!=null) && (subLevels.size()>0))
    {
      OrgViewLevelIF  subLevel;
      Boolean ret;
      Element xmlLevel;
      String name;
      for (int i=0;i<subLevels.size();i++)
      {
```

And create them	`xmlLevel = (Element)subLevels.get(i);`
	`// Add the sublevel to the level`
	`name = xmlLevel.getAttribute (nameTxt);`
	`subLevel = level.doAddOrgViewLevel (name, null);`

Create their sub-levels and associates their contracts	`// Then fills it`
	`ret = fillLevel (org, subLevel, xmlLevel, contractTxt, levelTxt, nameTxt);`
	`if ((ret== false)`
	`return  ret;`
	`}`
	`}`

Return TRUE to the caller	`return true;`
	`}`
	`%>`

Uploading a Modified Organization View

The steps are the same as creating a new cost structure except for the following:

- In Step 4, you load the cost structure and check the references
- As the cost structure already exists, you do not need to carry out Step 6

The sample JSP code shows how to perform the modified Step 4:

Loads the cost structure to update	`OrgViewIF   orgView;`
	`ObjectId orgViewId = ObjectId.instantiate (request.getParameter ("orgviewId"));`
	`orgView = (OrgViewIF)ObjectMgr.getOrgViewNode (orgViewId);`

Check the two references match	`// Test if the user reference already exists`
	`if (!userRef.equals(orgView.getUserReference()))`
	`{`
	`// It’s not the right cost structure`
	`Hashtable urlParameters = new Hashtable ();`
	`urlParameters.put ("ex_msg", jspHelper.localize ("wrong_user_reference", new Object[]{userRef, orgView.getUserReference()}));`
	
	`response.sendRedirect (jspHelper.encodeURLFunct ("GLOBAL.LOGIC_ERROR", urlParameters, true));`
	`return;`
	`}`

CHAPTER 6

Using the CustDim Loader

In This Section

About the CustDim Loader

The CustDim Loader is a set of components you use to quickly build an efficient process to load customer data.

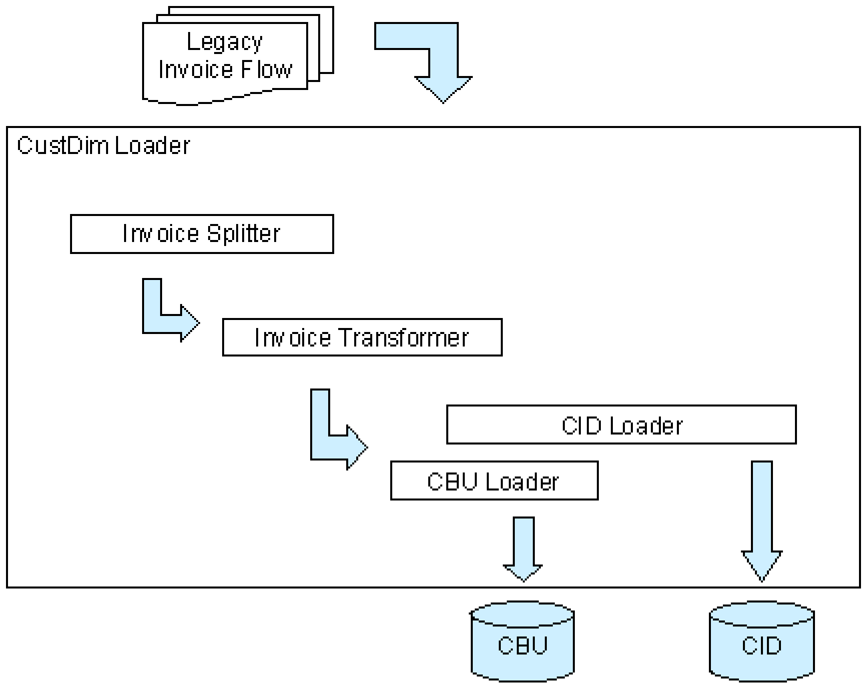
The main process of this loader is the following:

1. Extract customer and contract information from invoice files
2. Load the information into the CBU dimension tables
3. Load the information into the CID customer tables

The information the CustDim Loader loads includes:

- The customer (and hierarchy information if specified in invoice files)
- The customer's billing accounts
- The customer's contracts

The diagram below shows the main components of the CustDim loader and how the CustDim Loader works.



By default, all of these steps are included in the default implementation of the CustDim Loader.

The only requirement when using the CustDim loader is that information in an invoice must be stored in a unique file. Of course, many invoices can be stored in only one file and many files with many invoices can be sent to the process.

The first component is the Invoice Splitter. This component splits invoice input files in unitary invoices that can be treated independently. The splitter can also strip unessential information from these files to create smaller files to increase the performance of subsequent processing. By default, the splitter supports invoice files in XML. You can customize the splitter to handle other formats.

The Invoice Transformer is in charge of mapping information of an invoice in a legacy format to a standard XML event.

The CBU loader and the CID loader process this event to update the CBU customer dimensions and CID Customer tables at the same time.

For more information about the format of the standard event, refer to the *CustDim Loader Schema Reference Documentation.*

Overview of the CustDim Loader Components

The CustDim Loader uses the standard TSM ISF integration framework. The CustDim Loader is an Integration Logic Connector and is configured and customized using the Integration Logic Studio.

The CustDim Loader has several integration processes, each of these corresponding to a particular step in the processing of invoices. Each integration process is linked with the next one using asynchronous communication for optimum processing speed.

The sequence of processing allows the CustDim Loader to be extremely configurable, efficient and scalable which are key factors for high volume customer data loading.

The components of the CustDim Loader are:

- Invoice Flow Splitter
- Invoice Transformer
- CID Loader
- CBU Loader
- Counters

For information about the ISF and using the Integration Logic Studio, refer to *Developing Connectors*.

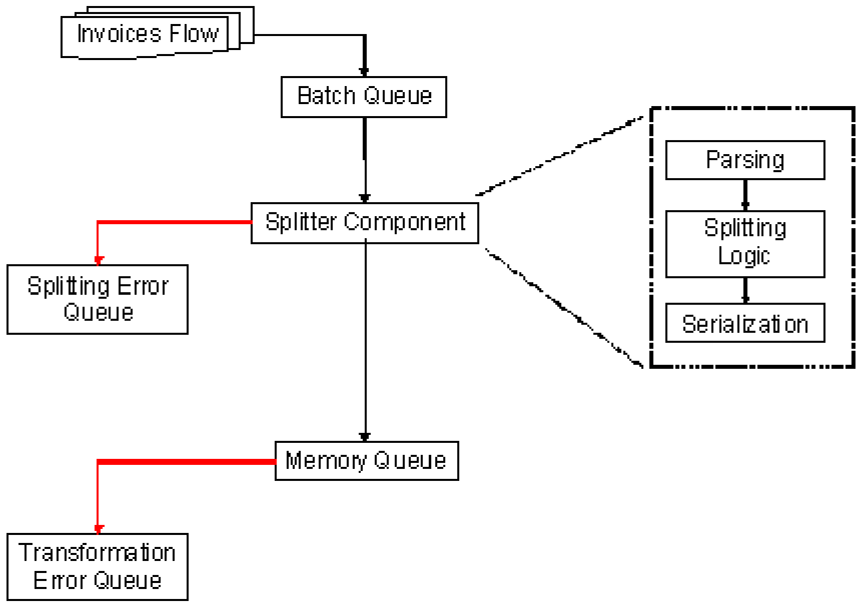
Invoice Flow Splitter

The Invoice Flow Splitter component is a Macro Runner processor.

This component:

- Reads the legacy invoice flow
- Selects and extracts the required invoice data
- Sends the invoice data to the next processing step

This diagram shows the steps of the integration process:



Process

Reading the invoice flow involves:

- Getting the list of invoice files to process
- Extracting the invoice files from a compressed archive
- Extracting the invoice data from a compressed invoice file
- Reading the invoice data from an invoice file
- In case of error, saving the invoice file for recovery

Selecting and extracting the required invoice data may consist of one or more of the following actions:

- Parsing the raw invoice data
- Filtering raw invoice data blocks
- Repeating raw invoice data blocks
- Building one or several XML legacy invoice message(s)
- Validating the generated invoice message with an XML Schema file
- In case of error, saving the invoice file for recovery

Sending the invoice data to the next processing step may consist of one or more of the following actions:

- Sending the legacy invoice message to the next integration process
- In case of error, saving the generated legacy invoice message for recovery in the transformation integration process

Components

- Inbound batch queue

  This component provides the following features:

  - Extract list of files to process from a batch list file
  - For each entry in this batch list, create an ISF message holding the file entry java stream
  - If the entry is a compressed archive, browse the archive and, for each archive entry, create an ISF message holding the compressed file java stream
- Splitter

  This component provides the following features:

  - Parse the invoice file holding the invoice XML data
  - Select invoice XML data tags that must be ignored/repeated

    - Build one or several invoice XML messages
- Outbound Memory queue

  This component provides the following features:

    - Allows two integration processes of the same connector to exchange asynchronously ISF messages through a shared memory space

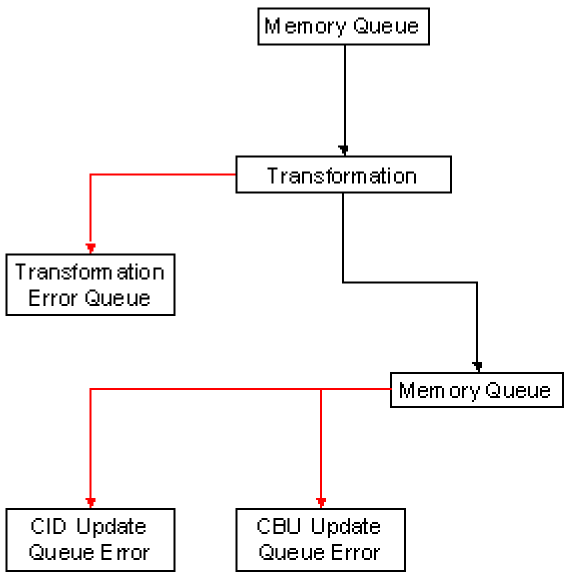
Invoice Transformer

The Invoice Transformer component is an Integration Process.

This component:

- Transforms the unitary invoice into a standard ISF invoice message
- Sends this message to the next processing step

This diagram shows the steps of the integration process:



Process

Transforming the unitary invoice may consist of one or more of the following actions:

- Mapping the legacy invoice XML message into a TSM invoice message using XSL technology
- Validating the TSM invoice message with an XML Schema file
- In case of error, saving the legacy invoice message for recovery

Sending TSM invoice message to the next processing step may consist of one or more of the following actions:

- Sending the TSM invoice message to the next integration process
- In case of error, saving the generated TSM invoice message for recovery in the CID and CBU Update integration processes

Components

The new processor types that might be used in this component include:

- XSL Transmapper

  This component provides the following features:

  - Apply XSL transformations on an XML Document (DOM or string format)
  - If several transformations are configured, this component creates an ISF message for each transformation and can buffer the output if required.
  - For each transformation, the resulting XML document can be validated using a XML Schema Definition file.
- Inbound and Outbound Memory queues

  These components provide the following features:

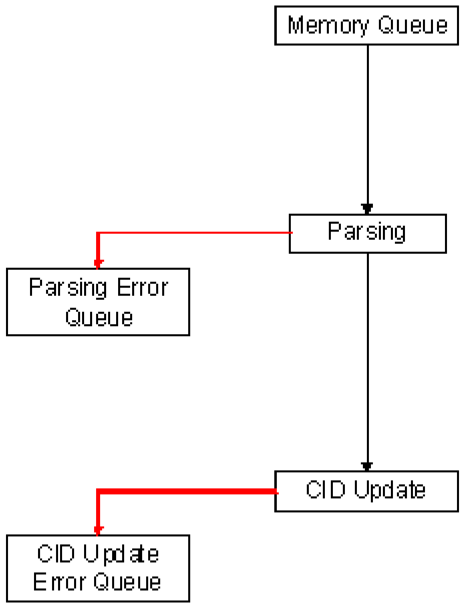
  - Allow two integration processes of the same connector to exchange asynchronously ISF messages through a shared memory space

CID Loader

The CID Loader Integration Process is a Script.

This component:

- Reads the TSM invoice message
- Updates the CID



▪

Process

Reading the TSM invoice message may consist of one or more in one of the following actions:

- Parsing the TSM invoice XML message
- Validating the TSM invoice XML message with an XML Schema file
- In case of error, saving the TSM invoice message for recovery

Updating the CID database may consist of one or more of the following actions:

- Update the CID using TSM APIs
- Increment statistical counters used to generate the statistical event (No. of contracts inserted, no. of organization inserted, etc…)
- In case of error, saving the TSM invoice message for recovery

As the CID Update component is implemented as a script, configuring the CID Update script may consist of one or more of the following actions:

- Customizing the script itself (Add/Remove script steps and transitions)
- Customizing the connector macros statements used by this script (Enable/Disable statistical counters)

Components

List of new processor types that might be used:

- XML Parser

  This component provides the following features:

  - Parse a XML Document in a string format and generates the corresponding DOM object
- XML Serializer
- This component provides the following features:
  - Serialize a XML Document in a DOM format and generates the corresponding string
- Inbound Memory queue

  This component provides the following features:

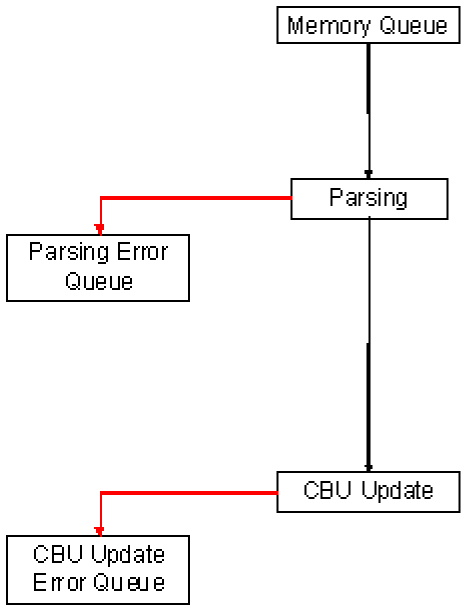
  - Allows two integration processes of the same connector to exchange ISF messages asynchronously through a shared memory space

CBU Loader

The CBU Loader Integration Process is a Script.

This component:

- Reads the TSM invoice message
- Updates the CBU



Process

Reading the TSM invoice message may consist of one or more of the following actions:

- Parsing the TSM invoice XML message
- Validating the TSM invoice XML message with an XML Schema file
- In case of error, saving the TSM invoice message for recovery

Updating the CBU database may consist of one or more of the following actions:

- Update the CBU using TSM APIs
- Increment statistical counters used to generate the statistical event (No. of contracts inserted, no. of organization inserted, etc…)
- In case of error, saving the TSM invoice message for recovery

As the CBU Update component is implemented as a script, configuring the CBU Update script may consist of one or more of the following actions:

- Customizing the script itself
- Customizing the connector macros statements used by this script (Enabling/Disabling statistical counters)

Components

List of new processor types that might be used:

- XML Parser. This component provides the following features:

  Parse a XML Document in a string format and generates the corresponding DOM object
- XML Serializer. This component provides the following features:

  Serialize a XML Document in a DOM format and generates the corresponding string
- Inbound Memory queue. This component provides the following features:

  Allows two integration processes of the same connector to exchange asynchronously ISF messages through a shared memory space

Counters

For each invoice flow coming from the backend and for each extracted unitary invoice extracted from this flow, the CustDim Loader generates statistical events periodically thus providing regular information on process status.

The Customer Dimension Loader connector is configured to generate statistical events, containing statistical counters values.

The following default counters are available:

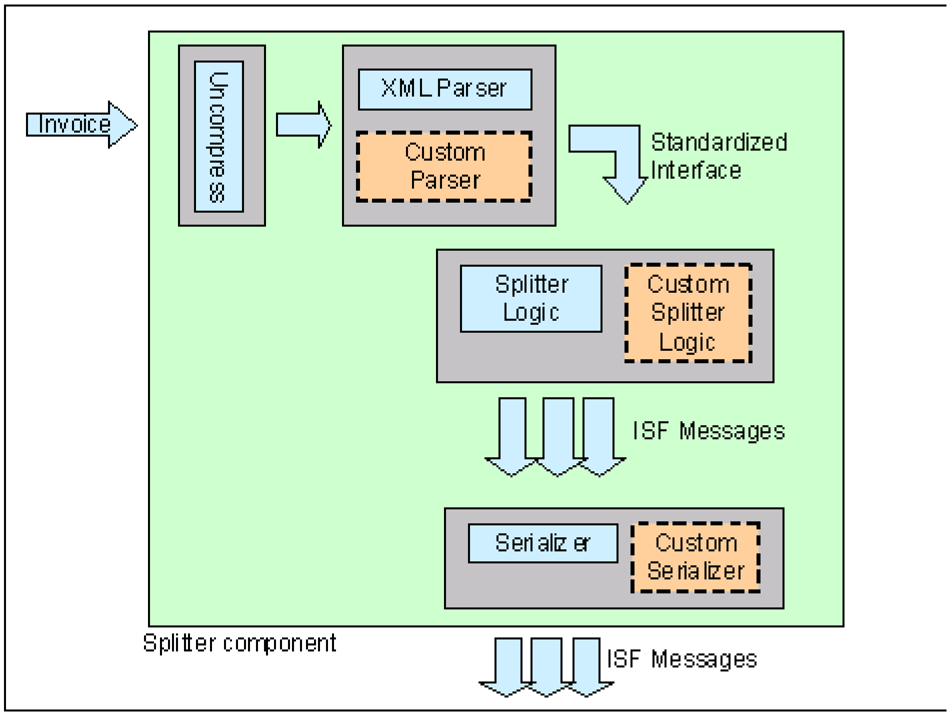
- Total number of entries in lists already or currently processed
- Total number of input files sent to splitter
- Total number of input files successfully split
- Total number of cut out invoices
- Total number of successfully transformed invoices
- Total number of successfully invoices inserted in CID
- Total number of successfully invoices inserted in CBU
- Total number of organizations inserted in CBU
- Total number of Levels inserted in CBU
- Total number of members inserted in CBU
- Total number of billing accounts inserted in CBU
- Total number of contracts inserted in CBU
- Total number of organizations inserted in CID
- Total number of Levels inserted in CID
- Total number of members inserted in CID
- Total number of billing accounts inserted in CID
- Total number of contracts inserted in CID

Each counter can be enabled or disabled, logged as part of an existing statistical group or as part of a new custom group

You can also add your own counters.

Working with Invoice Files

About Working with Invoice Files



To specify the list of invoice files to process

1 Create a new XML file with the same structure as this sample:

```
<batch_list>
   <directory_lists>
       <directory>c:/invoices/0101/</directory>
         <files>
          <file>invoices_1.xml</file>
          <file>invoices_1.xml</file>
         </files>
  </directory_lists>
</batch_list>
```

2 In the `<directory>` element, specify the full path of the directory holding your legacy invoice files.

3 For each invoice file to process, under `<files>` specify the name of you invoice file using the syntax:

`<file>invoices_filename.xml</file>`

This path must be relative to the path specified in the `<directory>` element.

4 For each invoice file specified in your batch list, you can specify additional parameters:

- These parameters will be attached to each ISF message corresponding to an invoice file.
- These parameters can be queried using the `Message.getUserValue` Script API.

Additionally, a parameter named `filename` is automatically inserted with the value set to the full file path.

To specify additional parameters, use the syntax as shown in this example.

The following example of an invoice list contains two parameters:

```
<batch_list>
   <parameters>
       <param1>value1</param1>
       <param1>value1</param1>
   </parameters>
   <directory_lists>
       <directory>c:/invoices/0101/</directory>
         <files>
          <file>invoices_1.xml</file>
          <file>invoices_1.xml</file>
         </files>
  </directory_lists>
</batch_list>
```

Processing Invoice Files in Non XML Format

If your invoice flow coming from the back end systems is not in XML, you need to customize the splitter component. this component is in charge of parsing the legacy invoice flow, filtering, and generating the unitary invoice.

You have to customize the parts of the splitter component that reads and parses the raw invoice data.

To process invoice files in non XML format

1 Write your own invoice parser in Java. Your java class must implement the `org.xml.sax.XMLReader` interface.

2 Open the Invoice Flow Splitting Integration Process.

3 Open the macro file that contains the macro namespace configured in your Macro Runner processor.

4 Locate the namespace and reference your custom parsing class instead of the standard parsing class.

Processing Invoice Files in Non Well Formatted XML

The invoice flow coming from the back end systems may be in XML but with a non hierarchical structure (flat XML). For instance, the legacy invoice flow may contain customer information followed by accounts and contracts instead of customer information, accounts, and contracts grouped under a common tag.

In this case, you may have to customize the splitter component which parses the legacy invoice flow, filters the information and generates the unitary invoice. You have to customize the filtering and splitting logic of the splitter component.

To process invoice files in non well formatted XML

1 Write your own invoice splitting logic in Java. Your java class must implement the `SplitterLogicIF` interface which extends the `org.xml.sax.ContentHandler` interface.

2 Open the Invoice Flow Splitting Integration Process.

3 Open the macro file that contains the macro namespace configured in your Macro Runner processor

4 Locate the namespace and reference your custom splitting logic class instead of the standard splitting logic class.

Processing Compressed Invoice Files

The inbound batch queue is designed to support the following:

- The batch list contains a list of one or more uncompressed legacy invoice files
- The batch list contains a list of one or more compressed legacy invoice files
- The batch list contains a list of one compressed file containing one or more nested compressed legacy invoice files

If you are using compressed invoice files which are not covered by the types of compression outlined above, you need to configure your CustDim Loader to process them. If you do not want to customize your CustDim Loader, you need to make sure your compressed invoice files correspond to one of the supported configurations.

Processing Invoice Files with Shared Information

If your legacy invoice files contain information common to several invoices, then you may have to repeat this information for each unitary generated invoice.

In this case, you may have to configure the splitter component which parses the legacy invoice flow, filters and generates the unitary invoice. You may have to configure the invoice tag to repeat.

To process invoice files with shared information

1. Open the Invoice Flow Splitting Integration Process.
2. Open the macro file that contains the macro namespace configured in your Macro Runner processor.
3. Locate the namespace and the `copyTag` macro variable
4. Set the value of this variable to the invoice tag to repeat

Pruning Invoice Files

If your legacy invoice files contain information that is not used to update the CID or CBU databases, then you can configure the splitter component to remove this information from the invoice flow. This information will not appear in the unitary invoice XML message.

You have to configure the splitter component to filter a specific element (open tag and corresponding end tag) from your raw legacy invoice data.

Either you prune the invoice just after parsing or when when generating the unitary invoice.

By pruning unnecessary data from your unitary invoice, you increase the performance of the CustDim loader by simply reducing the amount and size of information being handled to update the CID and CBU.

To prune invoice data after parsing

1. Open the Invoice Flow Splitting Integration Process.
2. Open the macro file that contains the macro namespace configured in your Macro Runner processor.
3. Locate the namespace and the `splitterFilter` macro variables.
4. Set the value of this variable to the name of the XML elements to remove. Use the syntax:

   ```
   parserFilters={"XML_element_name1","XML_element_name2",...}
   ```

   Make sure there is no space between element names.

   ---
5. Save your changes.

Example of pruning after parsing

In this example, you need to keep everything except the `<tag2.1>` element:

```
<tag1.1>

      <tag2.1>…</tag2.1>

      <tag2.2>…<tag2.2>

</tag1.1>

<tag1.2>…</tag1.2>
```

In your parser, set the parser Filter value to `tag2.2`

In this example, you need to keep everything except the `<tag2.1>` element and the `<tag2.3>` element:

```
<tag1.1>

      <tag2.1>…</tag2.1>

      <tag2.2>…<tag2.2>

      <tag2.3>…<tag2.3>

</tag1.1>

<tag1.2>…</tag1.2>
```

In your parser, set the parser Filter value to `tag2.2` and `tag2.3`.

To prune invoice data when generating the message

1. Open the Invoice Flow Splitting Integration Process.
2. Open the macro file that contains the macro namespace configured in your Macro Runner processor.

3. Locate the namespace and the `splitterFilter` macro variables.
4. Set the value of this variable to the name of the element to remove.
5. Repeat as necessary for each element to remove.

Example of pruning when generating an invoice message

In this example, you need to generate two unitary invoice messages:

- One message with everything except `tag2.1`
- One message with everything except `tag2.2`

```
<tag1.1>

      <tag2.1>…</tag2.1>

      <tag2.2>…<tag2.2>

</tag1.1>

<tag1.2>…</tag1.2>
```

You configure two splitter Filters, the first with the value set to `tag2.1`, and the second with value set to `tag2.2`.

In this example, you need to generate two unitary invoice messages:

- One message with everything except `tag2.1` and `tag2.2`
- One message with everything except `tag2.3`

```
<tag1.1>

      <tag2.1>…</tag2.1>

      <tag2.2>…<tag2.2>

      <tag2.3>…<tag2.3>

</tag1.1>

<tag1.2>…</tag1.2>
```

You configure two splitter Filters, the first with the value set to `tag2.1` and `tag2.2`, and the second with value set to `tag2.3`.

Customizing Invoice Mapping

Setting Input Date/Numeric Format

The unitary invoice message may contain date and numeric data in a specific format.

The CID and CBU loader must be able to parse and recognize this specific format so it can update the CID or CBU databases.

When your invoice data is using a specific format, you can specify the date and numeric format patterns in your connector macro file.

To specify the format, do the following:

1 Open your CustDim Loader Connector macro file.

2 Locate the Date and Numeric format macro expression.

3 Replace the default date and numeric patterns.

Setting the Object ID

In the `AddInvoice` event, every object can be identified by:

- A BizKey
- An UIDN

The CustDim Loader Invoice transformer maps these attributes to the values of the legacy invoice.

The BizKey is mandatory as it used to identify the object both in the CBU (BizKey column of the object) and in the CID (Object legacy ID).

The UIDN is not mandatory but should be set (at least for contracts and billing accounts) as it is used to reconcile loaded facts with dimension tables (contract and billing account).

The UIDN is a reference that is shared by the customer and the operator. If there is no specific value in the invoice files for this attribute, you can set it to the same value as the BizKey. The UIDN will be set in the UIDN column of the object in the CBU. By default, this attribute is only set in the CID for customer and level objects (reference attribute).

For more information, refer to the *CustDim Loader Schema Reference Documentation*.

Example:

The contract invoice legacy format specifies two attributes for every contract:

- The Contract reference number
- The Contract line number

A possible mapping is:

- Contract reference number mapped to the BizKey attribute (should never change)
- Contract reference number mapped to the UIDN attribute (you could also store the line number but advantage of reference number is stability)

By default, the line number is mapped to the `NAME` attribute.

Your choice must be consistent with the CID2CBU mapping policy. In the previous example, if you decide to map the Contract UIDN with the reference number, you must modify the CID2CBU mapping policy to set the UIDN value with the CID contract legacy value instead of the line number (default implementation).

For more information about customizing the CID2CBU loader, refer to *Customizing the BLM to CBU Mapping* in this manual.

Setting Reference Object Codes

Some objects have attributes that are related to reference objects in the CID. Some of the attributes are mandatory (org type for organization), others are optional (Rate plan for contract).

To specify the link with a CID reference object, you must set its code or its BIZ key (legacy id) attribute in the add invoice event.

Some reference objects in the CID have no code attribute. In this case, you must set the code or the BIZ key attribute with the BIZ key value (code attribute will be treated as BIZ key)

The method to set this value depends on the information specified in the legacy invoice.

- Setting the code value when not specified in the legacy invoice

  If the code attribute is mandatory, you must decide at design time the value of the code consistent with the associated reference object in the CID. Then you have to configure the Invoice Transformer to set this value as a constant.

  Example:

  The organization type is mandatory but not specified in the legacy invoice. You could decide that all organization are business and configure the transformer to set the value `BUSINESS` as a constant in the organization attribute `typecode`.

  In order to configure the Invoice Transformer to perform this processing, you can directly set the content of your field in your XSL style sheet by using `element`, `attribute` or `value-of` XSL statements.

- Setting the code value when specified in the legacy invoice and consistent with the CID

  If the object reference code value is specified in the legacy invoice, you can use the same code in the CID. In this case, you have just to configure a simple mapping in the Invoice Transformer.

  If you choice this solution, you will be dependent for reference objects code value of the legacy invoice format. When deploying self care features, you may have to review all reference object codes.

  In order to configure the invoice transformer to perform this processing, you can create a dedicated XSL template and then copy it in the destination element using `select`, `copy` or `copy-of` XSL statements.

- Setting the code value when specified in the legacy invoice and consistent with the CID

  If the object reference code value is specified in the legacy invoice but with a different codification than the one you want to use in the CID, you have to configure a look up mechanism in the transformer to replace the legacy code value by the CID reference object code attribute.

  In order to configure the Invoice Transformer to perform this processing, you have to configure a lookup mechanism in your XSL transformation.

  To specify your lookup table, you can do one of the following:

  - Store your lookup table in a separate file (for instance `lookuptable.xml`) referenced by the XSL transformation file (using an XSL statement like `select(Document('lookuptable.xml')/tabledata)`)
  - Use XML Namespaces and embed the lookup table in the XSL transformation file (using an XSL statement like "`select(Document('')/tabledata)`")

  To implement the lookup operation, you may:

  - Use an XSL variable to store your lookup table (either external or embedded)
  - Use a XSL key to index your lookup table
  - Use a XSL template to perform the object reference code translation

Refer to your favorite XSL reference for more information about working with XSL.

For more information about the structure of the XML event, refer to the *CustDim Loader Schema Reference Documentation*.

Setting Miscellaneous Object Attributes

- Setting Organization, level and member contact information in the CID

  In the CID, the following types of contacts can be used to characterize these objects:

  - legal
  - billing

  If this information is specified in the legacy invoice, you can set them in the legal contact and billing contact attributes of the related object in the add invoice event.

  Example:

  Tto set the organization company name in the CID, you must configure the Invoice Transformer to map the information in the legal contact company name of the organization in the add invoice event.

- Setting `IsPaymentResponsible` flag

  In the CID, you can specify if an organization, a level or a member is responsible for payment. If you do not want to specify this information or cannot obtain it from the invoice files, the CID loader automatically sets this value to 0 (No).

- Setting contract level type in the CID:

  In the CID, you can specify if a contract is global (use to declare global tariffs) or elementary (normal contract). If you do not want to specify this information or cannot obtain it from the invoice files, the CID loader automatically sets it to 'I' (normal contract).

- Setting contract Status change date:

  Every time the contract status changes, you can specify the date in the CID. If you do not want to specify this information or if you cannot obtain it from the invoice files, the CID loader automatically sets it to the current date if status has changed.

For more information, refer to the *CustDim Loader Schema Reference Documentation*.

Setting Custom Object Attributes

Each object comes with a set of core attributes. However, depending on your requirements, you may need to add custom attributes. You do this by declaring additional parameters in the CID and adding new columns in the CBU customer dimension tables.

The CustDim Loader supports the addition of the following attribute types:

- String
- Numeric (Numeric or Integer)
- Date

You can set custom attributes in the CID, in the CBU, or in both.

Setting custom object attributes involves:

1. In the Invoice Transformer, under the `CUSTOM` element of the object, enter:
   - An element identifying the custom attribute
   - The corresponding attribute value

For more information about:

- the Invoice Transformer, refer to the *Overview of the CustDim Loader Components* in this manual.

- using the ILS to edit your Invoice Transformer Integration Process, refer to *Developing Connectors*.

2 To set the custom attribute in the CBU:

- Add a new column in the table related to the object
- Configure the CBU loader to load this attribute in the created column

3 To set the custom attribute in the CID:

- Declare the parameter in the CID.

For more information about declaring parameters in the CID, refer to *Developing Telco Service Manager*

- Configure the CID loader to load this attribute as object additional information in the CID

For customer and level objects, if you want to add a custom attribute in the CBU and if you also activate notification for organization view management, you must also add it in the CID. If not, the value will be erased when an Update organization event is processed by the CID2CBU loader.

If you have loaded a custom attribute both in the CID and in the CBU and if you also activate notification for organization view management, you must customize the CID2CBU loader to synchronize the attribute in case of an update org notification.

To configure the CBU loader to load custom attributes

1 Open you CustDim Loader macro file.

2 Find the `CBU loader parameters` section

3 For the object having the custom attributes, enter the following:

- The name of the custom table column
- The Xpath to find the value of the attribute in the event XML file
- The attribute type

  To get the list and syntax of supported attribute types, look under the `Additional Type Parameters` section of the same file

Use the syntax:

```
<ObjectName>= {{"Column name","./CUSTOM/<attribute
code>,CustDimLoader.Type.<attribute type>}, {…},…}
```

Example

To declare the custom attribute `CUSTOM1` of type string in column `COLUMN1` of the Billing Account dimension table:

```
billingaccount = {{"COLUMN1","./CUSTOM/CUSTOM1",
CustDimLoader.Type.STRING}}
```

To configure the CID loader to load custom attributes

1 Open you CustDim Loader macro file.
2 Find the `CID loader parameters` section
3 For the object having the custom attributes, enter the following:
   - The name of the custom table column
   - The Xpath to find the value of the attribute in the event XML file

   Use the syntax:

   ```
   <ObjectName>= {{"Column name","./CUSTOM/<attribute code>},
   {…},…}
   ```

Example

To declare the custom attribute `CUSTOM1` of type string in additional parameter `PARAM1` of the Billing Account object:

```
billingaccount = {{"PARAM1","./CUSTOM/CUSTOM1"}}
```

Managing Billing Accounts as Actors

If you want to declare users at the billing account level (for security reasons), you must have a level (in the CID, an actor of type ‘L’ in the CBU) dedicated to this billing account. This can be not the exact customer model in the legacy invoice. In this case you have to configure the transformer to create a level between the legacy billing account parent level and the billing account.

You must set the Bizkey (and potentially the UIDN, name and legal contact company name) of this level with values set for the billing account.

To avoid merging these values with the ones of real levels, you could decide of a convention like: take the billing account Biz key, add “B/” at the beginning and set the value in the Level Biz key.

Example:

The legacy customer model supports multiple billing accounts below a customer. You want to manage security at billing account level. You must configure the transformer to create a “fake” level below the customer and attach the billing account to it.

In order to configure the invoice transformer to perform this processing, you can define a dedicated XSL template called by the customer XSL template that will create the “fake” level and insert it below the customer.

Refer to your favorite XSL reference for more information on how to work with templates.

Managing statistics

The statistics generated by the CustDim Loader connector can be customized in different ways:

- You can group counters values in one or several statistical events
- You can configure new counters
- You can remove existing counters
- You can define how statistical counters are logged into statistical events

In order to customize statistics, you have several options:

- You can enable or disable the specific counters of the CID and CBU loader processes
- You can insert or remove a counter processor at any place of your integration process.
- You can modify the content of the generated statistical events

For more information about:

- Working with counter processors, refer to *Developing Connectors*

- Working with statistics, refer to *Administrating Telco Analytics Manager*

Customizing CBU Name Attribute Mapping

For every object, you can set the NAME element of the add invoice event during the transforming phase. In this case, the CBU loader use it to set the value of the NAME column of the record.

If the NAME element is not, the CBU loader automatically applies the the following policy depending on the object type:

- Level

  Company name of the legal contact if filled in the input event

  “First name” “blank” “last name” of the legal contact otherwise (residential organization)
- Member

  “First name” “blank” “last name” of the legal contact (if filled in the input event)

Contract

Line number

Billing account

BizKey

The default policy is applied only if the NAME element is not specified. If you set the value to null in the event (<NAME></NAME>), The NAME column value will be set to NULL.

C H A P T E R 7

Creating Report Queries in SQL

In This Section

About Building Reports

The purpose of the CBU is to allow users to analyze their usage and bill details. In order to enable your application to do this, you can create reports with a reporting tool or even simple SQL. The basis of these reports is extract information from the CBU fact tables using the various dimensions as a selection criteria. To insure user confidentiality, reports must also implement security and prompt the user (User table).

For efficient queries and to ensure reasonable response times, reports should filter dimension tables on:

- Surrogate key
- Label
- Attributes of type

As you can use different reporting tools, the examples in this section shows the SQL query that can be used to obtain reports.

Writing reports involves:

- Writing a simple report query
- Applying the security to your query
- Writing advanced queries

Do not use the `T_NAME` logical type as filter criteria.

Your final reports must not display surrogate or business keys. These keys are for development or integration purposes only.

Writing Simple Report Queries

This example report lists the invoice details.

The details in the report are billed usage detail records of an invoice.

In this example, the report extracts information to display the following information for every selected billed usage detail record:

- Contract name (MSISDN if mobile contract)
- Date of the record
- Destination number
- Length of the call
- Type of tariff
- Usage fee

The first example does not implement security. If you use the first example as is, there are no restrictions on what the user can access. All users of the CBU can extract and view information for all of the customers in the CBU. To secure access to a fact table, you must always implement security in your reports. The second sample shows you how to implement security for the first report.

Simple Query

SQL STATEMENTS	COMMENTS
`SELECT`	Select the following:
`contract.CONTRACT_NAME,`	Contract name (MSISDN if mobile contract)
`budr.BUDR_START,`	Date of the record
`budr.DEST_NUMBER,`	Destination number
`budr.BUDR_VOLUME,`	Record volume (length of the call)
`tariff.UDR_TARIFF_NAME,`	Type of tariff using the name of the tariff table dimension
`budr.BUDR_FEE`	Usage fee

SQL STATEMENTS	COMMENTS

`FROM`
`NMY_BILL_PERIOD_DIM bp,`
`NMY_CONTRACT_DIM contract,`
`NMY_BILL_ACCT_DIM ba,`
`NMY_UDR_TARIFF_DIM tariff,`
`NMY_BUDR_FACT budr`
	This report uses the following tables:
	NMY_BILL_PERIOD_DIM to select a bill period
	NMY_CONTRACT_DIM to get the name of the contract
	NMY_BILL_ACCT_DIM to select the billing account
	NMY_UDR_TARIFF_DIM to get the tariff type
	NMY_BUDR_FACT fact table storing the billed usage detail records

`WHERE`
`( bp.BILL_PERIOD_KEY=budr.BILL_PERIOD_KEY )`
`AND( budr.BILL_ACCT_KEY=ba.BILL_ACCT_KEY )`
	Join to select only billed usage detail records related to the selected billing account and bill period

`AND(contract.CONTRACT_KEY=budr.CONTRACT_KEY )`	Join to get the contract name related to each usage record
`AND(budr.UDR_TARIFF_KEY=tariff.UDR_TARIFF_KEY )`	Join to get the tariff type related to each usage

`AND( bp.BILL_PERIOD_NAME = '&2' )`
`AND( ba.BILL_ACCT_UIDN = '&3' )`
	Prompt to enter the billing account and the bill period

`ORDER BY`
`1 ASC,`
`2 ASC`
	Order the usage by date then by contract.

Simple Query Including Security

SQL STATEMENTS	COMMENTS

`SELECT`
`contract.CONTRACT_NAME,`
`budr.BUDR_START,`
`budr.DEST_NUMBER,`
`budr.BUDR_VOLUME,`
`tariff.UDR_TARIFF_NAME,`
`budr.BUDR_FEE`
	Select the following:
	• Contract name.(MSISDN if mobile contract)
	• Date of the record
	• Destination number
	• Record volume (length of the call)
	• Type of tariff using the name of the tariff table dimension
	• Usage fee of the usage

`FROM`
`NMY_USER_DIM usr,`
`NMY_FUNCT_SCOPE_LNK fs_lnk,`
`NMY_ACTOR_LNK a_lnk,`
`NMY_CONTRACT_ACL acl,`
	To implement security you need usage of the following tables:
	NMY_USER_DIM to select a specific user of the system
	NMY_FUNCT_SCOPE_LNK to select the scope of the user for this report
	NMY_ACTOR_LNK to select actors in the scope of the user for this report
	NMY_CONTRACT_ACL to secure access to the usage fact table to only records related to authorized contracts.
	Note that the ACL table used depends on the fact table you want to access. Use the NMY_BILL_ACCT_ACL table if your report is about main invoice fact table

SQL STATEMENTS	COMMENTS

`NMY_BILL_PERIOD_DIM bp,`
`NMY_CONTRACT_DIM contract,`
`NMY_BILL_ACCT_DIM ba,`
`NMY_UDR_TARIFF_DIM tariff,`
`NMY_BUDR_FACT budr`
	• NMY_BILL_PERIOD_DIM to select a bill period
	• NMY_CONTRACT_DIM to get the name of the contract
	• NMY_BILL_ACCT_DIM to select the billing account
	• NMY_UDR_TARIFF_DIM to get the tariff type
	• NMY_BUDR_FACT fact table storing the billed usage detail records

`WHERE`
`( bp.BILL_PERIOD_KEY=budr.BILL_PERIOD_KEY )`
`AND( budr.BILL_ACCT_KEY=ba.BILL_ACCT_KEY )`
`AND(contract.CONTRACT_KEY=budr.CONTRACT_KEY )`
`AND(budr.UDR_TARIFF_KEY=tariff.UDR_TARIFF_KEY )`
	Join to select only billed usage detail records related to the selected billing account and bill period
	Join to get the contract name related to each usage record
	Join to get the tariff type related to each usage

`AND(fs_lnk.USER_KEY = usr.USER_KEY )`	Join to get the scope of the selected user

`AND(fs_lnk.ACTOR_KEY = a_lnk.PARENT_ACTOR_KEY )`
`AND(a_lnk.CHILD_ACTOR_KEY = acl.ACTOR_KEY )`
	Join to get all actors visible for the user scope

`AND( acl.CONTRACT_KEY = contract.CONTRACT_KEY )`	Join to get only authorized contracts
`AND( fs_lnk.FUNCT_CODE = 'USAGE' )`	Filter the scope to the functional code consistent with the purpose of the report
`AND( usr.USER_LOGIN =  '&1' )`	Prompt to enter the user login

`AND( bp.BILL_PERIOD_NAME = '&2' )`
`AND( ba.BILL_ACCT_UIDN  =  '&3' )`
	Prompt to enter the billing account and the bill period

`ORDER BY`
`1 ASC,`
`2 ASC`
	Order the usage by date then by contract.

In these reports, users with authorized roles can select contracts both belonging to their level and owned by them.

Writing Advanced Report Queries

The following reports sums the fees, the number of billed usage detailed records and the number of related contracts associated with a specific cost center or to one of its children for a specific time period.

In this example, the report extracts information to display the following information:

- Cost center name
- Cost center full path name in the cost center hierarchy
- Number of contracts for each cost center
- Number of usage records for each cost center
- Sum of usage records fee for each cost center

Advanced Query

SQL STATEMENTS	COMMENTS

`SELECT`
` cc_lnk.RANK_FROM_PARENT rank,`
` child.COST_CENTER_PATH path,`
` child.COST_CENTER_NAME name,`
` count(distinct contract.CONTRACT_UIDN),`
` count(budr.BUDR_FEE) count,`
` sum(budr.BUDR_FEE) fee,`
	Select the following
	• cost center rank from is parent for ordering
	• complete cost center path in cost center hierarchy
	• cost center name
	• count of number of contracts for each cost center
	• count of number of usage records for each cost center
	• sum usage records fee for each cost center

`FROM`
` NMY_USER_DIM usr,`
` NMY_FUNCT_SCOPE_LNK fs_lnk,`
` NMY_ACTOR_LNK a_lnk,`
` NMY_ACTOR_DIM actor,`
` NMY_CONTRACT_ACL acl,`
	Implement security as in the preceeding example
	Note that in this example we also get the actor dim to select only contracts that belong to a level and not the ones owned by the user

` NMY_COST_CENTER_DIM child,`
` NMY_COST_CENTER_LNK cc_lnk,`
` NMY_CONTRACT_DIM contract,`
` NMY_CC_CONTRACT_LNK ccc,`
	To use cost centers in your reports, you must use the cost center dimension table and the link tables between cost center and:
	Cost center hierarchy: NMY_COST_CENTER_LNK
	Contracts: NMY_CC_CONTRACT_LNK

` NMY_DATE_DIM d,`	You need to use this table to filter usage records BY time period
` NMY_BUDR_FACT budr`	Fact table storing the billed usage detail records

SQL STATEMENTS	COMMENTS

`WHERE`
` ( usr.USER_LOGIN =  '&login' )`
` AND  ( fs_lnk.USER_KEY = usr.USER_KEY )`
` AND  ( fs_lnk.FUNCT_CODE = 'USAGE' )`
` AND  ( fs_lnk.ACTOR_KEY = a_lnk.PARENT_ACTOR_KEY )`
` AND  ( actor.ACTOR_KEY = fs_lnk.ACTOR_KEY )`
` AND  ( actor.ACTOR_TYPE = 'L' )`
` AND  ( a_lnk.CHILD_ACTOR_KEY = acl.ACTOR_KEY )`
` AND  ( acl.CONTRACT_KEY = contract.CONTRACT_KEY )`
	Join to apply security and get only visible contracts for the user level as in previous example.
	Note the restriction on the actor type to select only the user's level.

` AND  ( cc_lnk.PARENT_CC_KEY = '&cck' )`	Select the root cost center to analyze
` AND ( cc_lnk.CHILD_CC_KEY = child.COST_CENTER_KEY  )`	Select the whole cost center hierarchy

` AND ( ccc.COST_CENTER_KEY = cc_lnk.CHILD_CC_KEY  )`
` AND ( ccc.CONTRACT_KEY = contract.CONTRACT_KEY  )`
	Get all contracts belonging to the cost center hierarchy

` AND ( contract.CONTRACT_KEY = budr.CONTRACT_KEY  )`
` AND ( budr.START_DATE_KEY = d.DATE_KEY)`
` AND ( d.DAY_DATE BETWEEN &first AND &last )`
	Select related usage records and restrict them by their date

`GROUP BY`
` RANK_FROM_PARENT,`
` COST_CENTER_PATH,`
` COST_CENTER_NAME`
`ORDER BY`
` RANK_FROM_PARENT;`
	Group the result by cost center of the cost centers hierarchy and order them by their rank from parents

CHAPTER 8

Deploying with Account Management Features

If you deploy self-care features, you should develop a connector with a backend system to get customer information in ‘real’ time.

Main issue concerns the validity of the customer dimensions Legacy id (BizKey in the CBU).

There are few chances that legacy ids based on the invoice files will be the same that the ones used to communicate with the backend system if this one is different that the one producing invoices.

In this case you should have to develop a migration process to make all legacy id and Biz key consistent with the new integrated backend system.

For the following objects, you must identify if the current legacy id can be used to communicate with the backend system:

- Organization (and level)
- Member
- Contract
- Billing account

- If you need to modify the legacy id you must:
- Extract from backend systems a conversion matrix between current legacy id and new Backend legacy Id
- Develop a batch to replace in the CID the old legacy id by the new one.
- Remove from the CBU all actors and ACL
- Reset to Null the Biz key value of all contracts and billing accounts
- Force Notification of all organizations

- Notify them: the Cid2Cbu loader will automatically resynchronize all customer dimensions with the correct BizKey

Note: Billing account specificity: A billing account has no UIDN in the CID. By default the Cid2Cbu loader set the billing account UIDN in the CBU with the billing account legacy id.

If the new legacy id is different from the UIDN already set in the billing account, you must:

- Declare an additional parameter in the CID to store the old billing account legacy id
- Customize the Cid2Cbu loader to synchronize the Billing account UIDN with this parameter.

Note: Contract specificity: A contract has no UIDN in the CID. By default the Cid2Cbu loader set the contract UIDN in the CBU with the contract line number.

If you have decide to map the contract UIDN in CBU with the BizKey (legacy id in the CID) and not with the line number and if the new legacy id doesn't match with the old one, you must:

- Declare an additional parameter in the CID to store the old contract legacy id
- Customize the Cid2Cbu loader to synchronize the contract UIDN with this parameter.

Note: Reference objects migration: You must check validity of the legacy id of every already declared reference objects and replace them if necessary to be able to communicate with the backend system.

About Deploying with Telco Service Manager

If your solution deploys Account Management features along with Telco Analytics Manager, you should develop a connector with a backend system to get customer information in real time.

When your Telco Analytics Manager and Telco Service Manager work together, one main concern is the validity of the customer dimension Legacy ID (BizKey in the CBU).

This is because the legacy IDs based on the invoice files are most likely not the same as the ones used to communicate with the backend systems. This is especially true when this system is different than the one producing invoices. In this case, you should develop a migration process to make all legacy IDs and Biz keys consistent with the newly integrated backend system.

For the following objects, you must specify if the current legacy ID can be used to communicate with the backend system:

- Organization (and level)
- Member
- Contract
- Billing account

To have coherent legacy IDs

1. Create a conversion matrix between current legacy ID and new Backend legacy Id
2. Develop a batch to replace the old legacy ID with the new one in the CID.
3. Remove all actors and ACL from the CBU
4. Reset the Biz key value of all contracts and billing accounts to `Null`
5. Force Notification of all organizations
6. Notify them: the CID2CBU loader will automatically resynchronize all customer dimensions with the correct BizKey

For Billing accounts

They have no UIDN in the CID. By default the CID2CBU loader sets the billing account UIDN to the billing account legacy ID in the CBU.

If the new legacy ID is different than the UIDN already set for the billing account, you must:

1. Declare an additional parameter in the CID to store the old billing account legacy ID
2. Customize the CID2CBU loader to synchronize the Billing account UIDN with this parameter.

For Contracts

They have no UIDN in the CID. By default the CID2CBU loader sets the contract UIDN with the contract line number in the CBU.

If you have decIDe to map the contract UIDN in CBU with the BizKey (legacy ID in the CID) and not with the line number and if the new legacy ID does not match with the old one, you must:

- Declare an additional parameter in the CID to store the old contract legacy ID
- Customize the CID2CBU loader to synchronize the contract UIDN with this parameter.

When migrating Reference objects, you must check valIDity of the legacy ID of every declared reference objects and replace them if necessary.

Index