Developer's Overview Guide for Siebel Communications Billing Analytics

Version: 5.1.1

Pub Date: 09/30/2006

Contents

1 Communications Billing Analytics (CBA) SDK

Introduction

This guide describes how to use CBA as a development platform and customize it for a particular deployment.

CBA includes three major components:

- Reporting
- Hierarchy
- Cost management

The Reporting and Hierarchy components have their own development guides (the "*Hierarchy Developer's Guide*" and the *"Reporting Developer's Guide"*) because each has a set of rich features.

This guide focuses on overall CBA development and refers to the other two guides as necessary.

2 CBA Architecture Overview

Overall Architecture of CBA

CBA includes two applications:

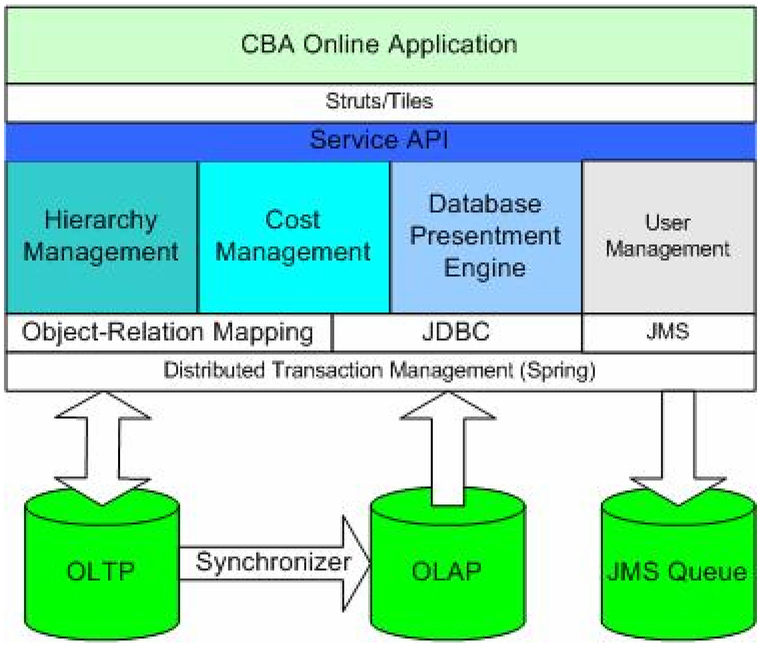
- **The online application**
- **Command Center**

Each application is packaged as one ear file:

- The online application ear is called **ear-tam-tbm.ear**.
- The Command Center application ear is called **estatement.ear**.

CBA Online Application

The online application is used by the end users. This diagram shows the overall architecture of the online application:



The user management module is used to authenticate and authorize a user. After the user has been authenticated, the user can then access different CBA feature such as hierarchy, cost management, reports, etc. See the CBA “*Application Guide”* for the details.

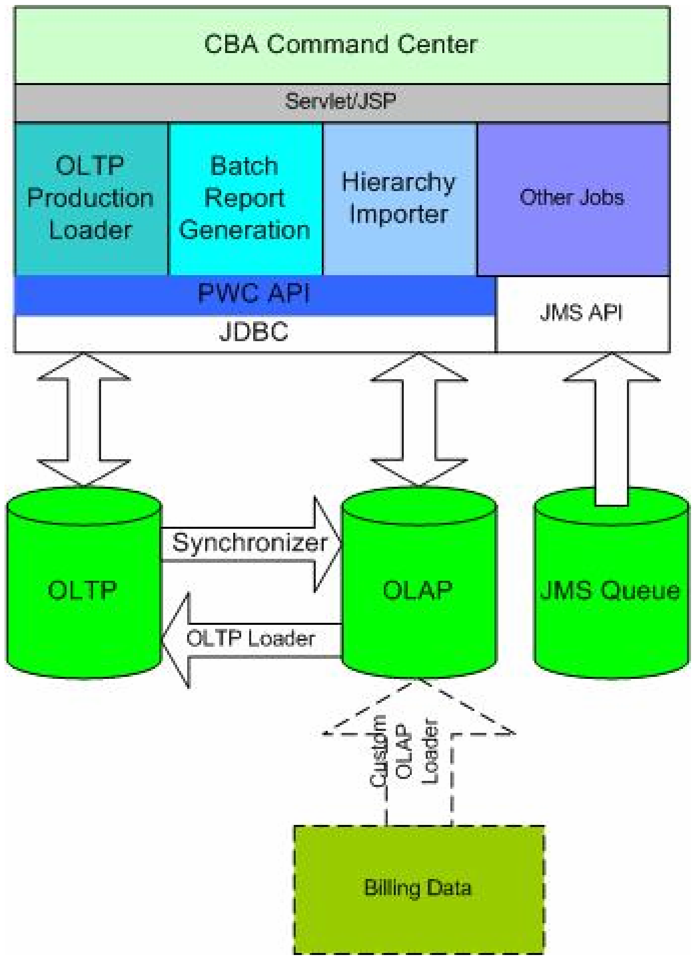
All the CBA online application UIs are built upon Struts/Tiles. The Struts actions talk with the Service APIs which then access different modules such as Hierarchy or Reporting.

- **Hierarchy Management Module**: Manages the life cycle of a hierarchy: creation, modification, expiration, deletion, copy, move, search, hierarchy-based access control (HBAC), etc.
- **Cost Management Module**: A set of tools to manage billing cost, such as re-bill, cost reallocation, and budget management.
- **Database Presentment Engine**: A framework to retrieve data from different data sources and present them as HTML, XML, or CSV. Offers features like paging, sorting, charting, bread-crumb, batch report, custom report, etc.
- **User Management**: A framework to authenticate a user and role based access control (RBAC).
- **DB Access**: The majority of our code uses Hibernate, which is an O-R mapping tool, to access OLTP database. The access to OLAP is through direct JDBC call to boost performance.
- **Transaction Management**: CBA uses distributed transaction (XA or JTA transaction) to manage database access and JMS access. The transaction is managed through the Spring Framework. The Spring framework is also used to manage object creations, etc.
- **OLTP Database**: The CBA transactional database, which includes transaction data such as user, account, services, and hierarchies.
- **OLAP Database**: The CBA non-transaction database, which includes billing data. It is a star-schema based data warehouse and includes dimensional tables, fact tables, and hierarchy tables.
- **OLTP-OLAP Synchronizer**: A process which synchronizes the information from OLTP to OLAP database. Currently, the main information being synchronized is the hierarchy. Any change made to hierarchy on the OLTP side will be synchronized at OLAP. The OLAP hierarchy schema is specially designed for queue performance and different from the OLTP hierarchy schema. However, they have the exact the same content - even the hierarchy node IDs are the same.
- **JMS Queue**: Currently, JMS queue is used to hold the batch report requests. The CBA online application can send a batch report to the JMS queue and later the request can be processed by CBA Command Center jobs.

CBA Command Center

The CBA Command Center is another application, and is packaged as a separate ear and deployed on a separate application server. It is typically used by an administrator to run batch jobs.

This diagram shows the overall architecture of the CBA Command Center:



(Note: The boxes with dotted lines are not in the product).

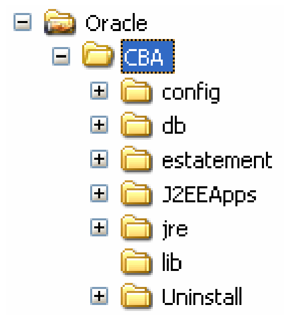
- **Command Center UI**: The UI is based on Servlet/JSP technology, not struts/tiles.
- **Jobs**: A Command Center job is a process which can be scheduled and run though the Command Center console. A job consists of one or more tasks. Each task is implemented as an EJB and has its own configuration parameters which can be configured through the Command Center UI. When a job runs, the tasks inside each job are invoked sequentially. Each task performs its task-specific processing.

  There is a set of pre-defined CBA jobs, such as OLTP loader, batch generator, hierarchy importer, etc. You must configure these jobs and scheduled them to run.
- **PWC API**: A set of APIs used to manage jobs.
- **JMS Queue**: The Batch Report job retrieves batch report requests from the queue then generates batch reports.
- **OLTP Loader**: Loads billing information into OLTP table. The default loader configuration loads companies, accounts, and service agreements from corresponding OLAP dim tables into OLTP tables and hierarchies from the exchange table into hierarchy tables. The OLAP dim tables and OTLP exchange table are usually populated by the OLAP loader.

- **OLAP Loader**: OLAP loader loads billing information into OLAP dim tables, fact tables, and OLTP exchange table. Note the current CBA product does not have an OLAP loader comes out of box and it is all custom effort.

CBA Package Structure

After you install CBA, the installed product directories are as shown here (this example is based on a Windows installation and installed on C: driver):



C:\Oracle\CBA

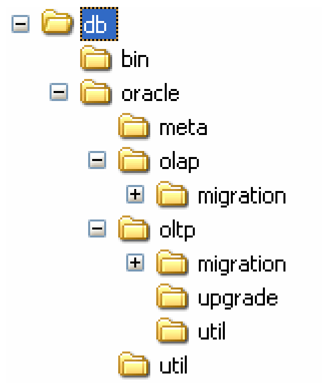
This is the default CBA installation directory on Windows. This directory is commonly referred to as EDX_HOME, which is set during CBA installation. See the *"Installation Guide"* for details.

C:\Oracle\CBA\config

This directory includes some configuration files for CBA. Currently, only log4j.xml is used by CBA and this directory is put on the class-path of CBA online application.

C:\Oracle\CBA\db

This directory includes CBA related databases scripts:



C:\Oracle\CBA\db\oracle: Contains the CBA DB schema for the Oracle database.

C:\Oracle\CBA\db\oracle\meta: Contains CBA meta data information. Not used.

C:\Oracle\CBA\db\oracle\olap: Contains CBA OLAP database schema.

C:\Oracle\CBA\db\oracle\olap\migration: Contains the OLAP schema migration scripts.

C:\Oracle\CBA\db\oracle\oltp: Contains CBA OLTP database schema.

C:\Oracle\CBA\db\oracle\oltp\migration: Contains the OLTP schema migration scripts.

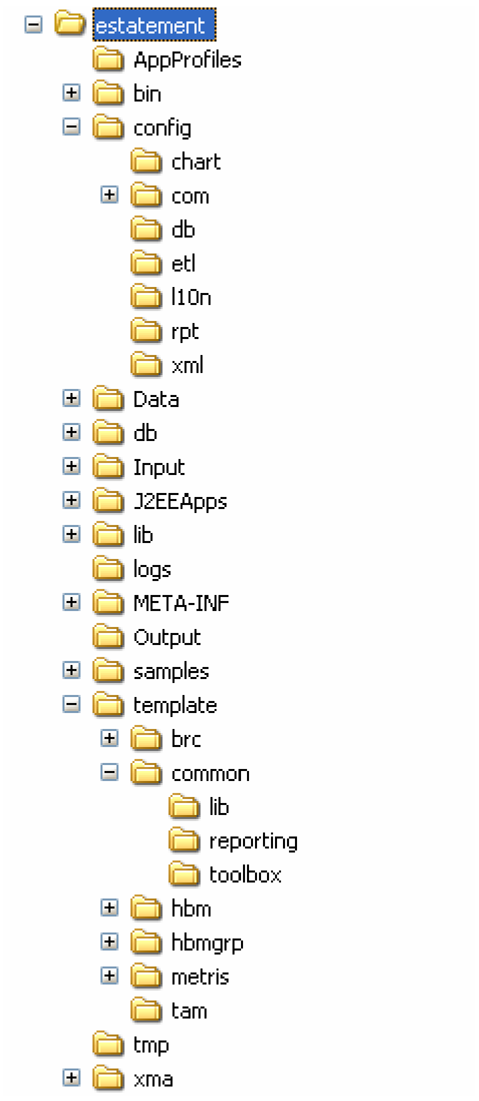
C:\Oracle\CBA\db\oracle\oltp\util: Contains OLTP utility scripts.

C:\Oracle\CBA\db\oracle\oltp\upgrade: Not used.

C:\Oracle\CBA\db\oracle\util: utility scripts for CBA

C:\Oracle\CBA\estatement

This directory contains information needed to run the CBA Command Center. This directory is commonly referred as "EDX_HOME". Note: This directory largely inherits from a legacy eBilling product and may contain contents not used by CBA.



```
⊟ estatement
    AppProfiles
  ⊞ bin
  ⊟ config
      chart
    ⊞ com
      db
      etl
      l10n
      rpt
      xml
  ⊞ Data
  ⊞ db
  ⊞ Input
  ⊞ J2EEApps
  ⊞ lib
    logs
  ⊞ META-INF
    Output
  ⊞ samples
  ⊟ template
    ⊞ brc
    ⊟ common
        lib
        reporting
        toolbox
    ⊞ hbm
    ⊞ hbmgrp
    ⊞ metris
      tam
    tmp
  ⊞ xma
```

C:\Oracle\CBA\estatement\AppProfiles: Not used by CBA.

C:\Oracle\CBA\estatement\bin: includes scripts to run CBA Command Center scheduler.

C:\Oracle\CBA\estatement\config: this directory contains various CBA configuration files. This directory should be put on the class-path of the CBA command center server. The log4j.xml in this directory is used for the CBA command center logging.

C:\Oracle\CBA\estatement\config\chart: includes the CBA report chart property files.

C:\Oracle\CBA\estatement\config\com: not used

C:\Oracle\CBA\estatement\config\db: not used by CBA.

C:\Oracle\CBA\estatement\config\etl: configuration files used for CBA OLTPProductionLoader job.

C:\Oracle\CBA\estatement\config\l10: Java resource bundle files for CBA reports.

C:\Oracle\CBA\estatement\config\rpt: CBA report XML definition files.

C:\Oracle\CBA\estatement\config\xml: contains XML schemas for CBA hierarchy download and input.

C:\Oracle\CBA\estatement\data: Not used.

C:\Oracle\CBA\estatement\db: Not used.

C:\Oracle\CBA\estatement\input: Not used.

C:\Oracle\CBA\estatement\J2EEApps: Not used.

C:\Oracle\CBA\estatement\lib: Not used.

C:\Oracle\CBA\estatement\logs: Not used.

C:\Oracle\CBA\estatement\META-INF: Not used.

C:\Oracle\CBA\estatement\Output: Output directory for CBA batch reports.

C:\Oracle\CBA\estatement\samples: Not used.

C:\Oracle\CBA\estatement\template: Contains CBA report Velocity templates.

C:\Oracle\CBA\estatement\tempalte\common\lib: Contains the VM libraries used by CBA.

C:\Oracle\CBA\estatement\template\common\reporting: Contains VM templates used by CBA reports.

C:\Oracle\CBA\estatement\template\tam: Not used.

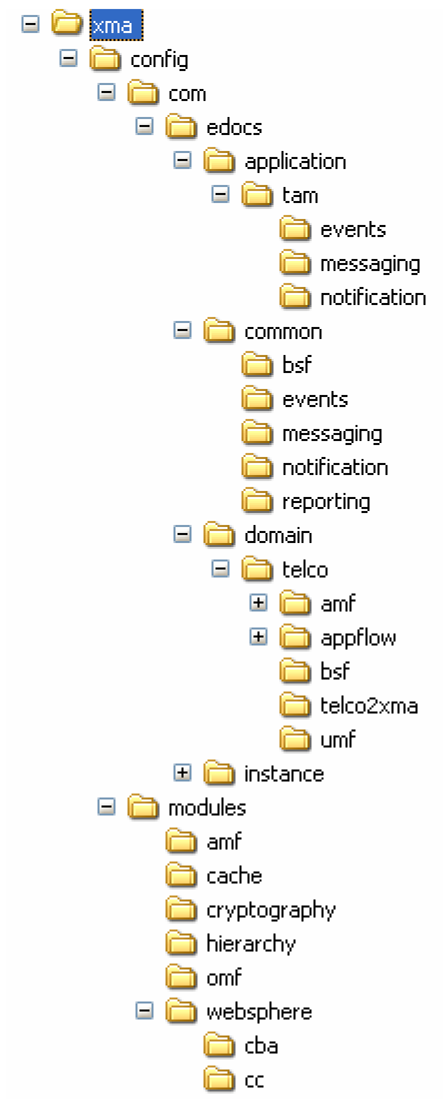
C:\Oracle\CBA\estatement\template\hbm: Not used.

C:\Oracle\CBA\estatement\template\hbmgrp: Not used.

C:\Oracle\CBA\estatement\template\metris: Not used.

C:\Oracle\CBA\estatement\tmp: Not used.

C:\Oracle\CBA\estatement\xma: One of the main configuration directories for CBA. This directory includes the Spring Framework-related configuration files.



```
⊟ xma
   ⊟ config
      ⊟ com
         ⊟ edocs
            ⊟ application
               ⊟ tam
                     events
                     messaging
                     notification
            ⊟ common
                  bsf
                  events
                  messaging
                  notification
                  reporting
            ⊟ domain
               ⊟ telco
                  ⊞ amf
                  ⊞ appflow
                     bsf
                     telco2xma
                     umf
            ⊞ instance
      ⊟ modules
            amf
            cache
            cryptography
            hierarchy
            omf
         ⊟ websphere
               cba
               cc
```

CBA maintains a hierarchy of Spring bean definition files. A bean can be defined in multiple Spring files and the order of loading is defined in repos-config.xml of the ear. If a bean is defined in more than one file, the subsequent bean definition will overwrite previous one. Note, you will find a file with same name in <EDX_HOME> but that file is not used. Instead, this file is here just for the purpose of illustration. This is the loading order of Spring files defined in repos-config.xml:

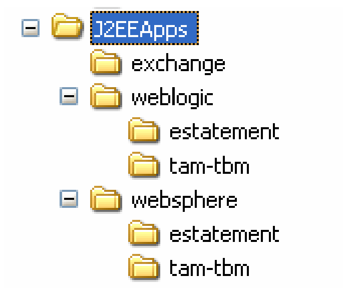
```
<value>modules</value>
<value>com/edocs/common</value>
<value>com/edocs/domain/telco</value>
<value>com/edocs/application/tam</value>
```

For WebSphere, some of the Spring files have been re-configured and they are saved in modules/websphere directory.

The “modules/websphere/cba” includes files used by the CBA online application for WebSphere. The “modules/websphere/cc” includes files used by the CBA Command Center application for WebSphere. The repos-config.xml in ear files for both applications have been modified to load respective WebSphere configuration files at the end.

C:\Oracle\CBA\J2EEApps

Contains the ear files for CBA.



C:\Oracle\CBA\J2EEApps\exchange: Contains a set of sample XML file for Hierarchy Import/Download.

C:\Oracle\CBA\J2EEApps\weblogic\estatement: Contains the Command Center ear file for WebLogic server.

C:\Oracle\CBA\J2EEApps\weblogic\tam-tbm: Contains the online application ear file for WebLogic server.

C:\Oracle\CBA\J2EEApps\websphere\estatement: Contains the Command Center ear file for WebSphere server.

C:\Oracle\CBA\J2EEApps\websphere\tam-tbm: contains the online application ear file for WebSphere server.

C:\Oracle\CBA\lib

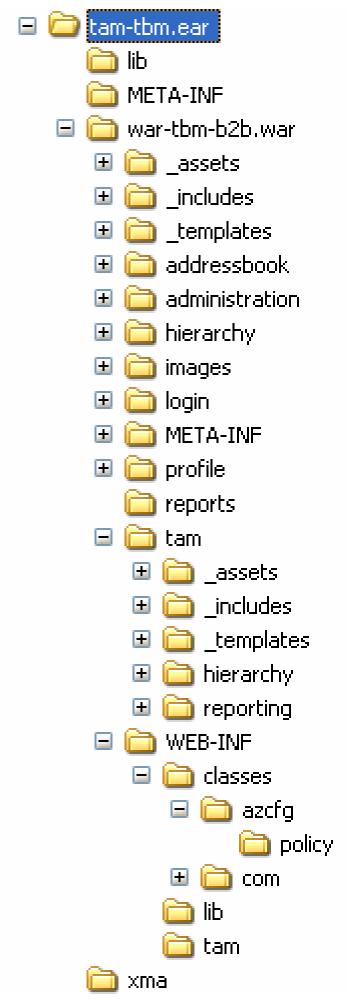
Contains lib files used by CBA.

CBA EAR Structure

This section describes what is packaged inside CBA ear files.

Ear-tam-tbm.ear

Ear-tam-tbm.ear is the CBA online application ear. In general, this ear is deployed in a cluster environment for the purpose of failover and load balance.



```
tam-tbm.ear
    lib
    META-INF
    war-tbm-b2b.war
        _assets
        _includes
        _templates
        addressbook
        administration
        hierarchy
        images
        login
        META-INF
        profile
        reports
        tam
            _assets
            _includes
            _templates
            hierarchy
            reporting
        WEB-INF
            classes
                azcfg
                    policy
                com
            lib
            tam
    xma
```

tam-tbm.ear: This is the root directory and contains EJB jar files.

tam-tbm.ear/lib: This directory contains the list of third-party lib files used by CBA.

tam-tbm.ear/META-INF: J2EE META-INF directory.

tam-tbm.ear/war-tbm-b2b.war: This is the CBA online application war file.

tam-tbm.ear/war-tbm-b2b.war: This directory contains a list of sub-directories. Most of those directories are inherited from another legacy eBilling application and are not used directly by CBA (except the tam and WEB-INF sub-directories).

tam-tbm.ear/war-tbm-b2b.war/tam: Includes CBA-related WEB components. Note: This directory is defined as a Struts module “tam” in the main struts-config.xml and the actual CBA action classes are defined in the strus-config-tam.xml file. Because of this, the URL accessing these actions should all prefix with “tam”.

tam-tbm.ear/war-tbm-b2b.war/tam/_assetts: Contains the images, JavaScripts and CSS files used by CBA.

tam-tbm.ear/war-tbm-b2b.war/tam/_includes and _templates directories: Contains JSP page fragments used by the application; many of these are tiles.

tam-tbm.ear / war-tbm-b2b.war / tam / hierarchy: CBA Hierarchy-related JSP pages.

tam-tbm.ear / war-tbm-b2b.war / tam / reporting: CBA Reporting-related JSP pages.

tam-tbm.ear / war-tbm-b2b.war / WEB-INF: J2EE WAR file WEB-INF directory.

tam-tbm.ear / war-tbm-b2b.war / WEB-INF / classes / azcfg / policy: Contains the CBA RBAC policy file.

tam-tbm.ear / war-tbm-b2b.war / WEB-INF / classes / lib: Includes libraries used by the war file.

tam-tbm.ear / war-tbm-b2b.war / WEB-INF / classes / com: Not used.

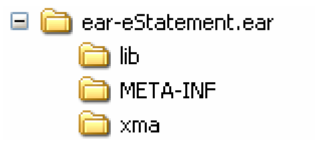
tam-tbm.ear / war-tbm-b2b.war / WEB-INF / classes / tam: Defines the struts validation files used by CBA. Not used.

tam-tbm.ear / xma: This directory contains a list of internal library files used by CBA. In this directory there is one jar file called api-<version-number>.jar. This jar file has all the public CBA APIs defined and also contains the JavaDoc html files.

Ear-estatement.ear

Ear-estatement.ear is used for the Command Center application. You should deploy the Command Center on a separate application server.

In general, you are not expected to modify this ear file during deployment.



Under ear-eStatement.ear directory are EJB and war files:

ear-eStatement.ear / lib: Third-party libraries.

ear-eStatement.ear / META-INF: J2EE META-INF directory.

ear-eStatement.ear / xma: Internal CBA libraries.

3 CBA Customization Guidelines

Overview

This chapter describes how to customize CBA. As always, customization of the product may bring challenges for migration. See the chapter on migrating CBA for details.

General Rules

CBA provides a set of core functionalities, such as reporting and hierarchy management, and a reference user interface (UI) to demonstrate these functionalities. The contract between CBA core and the UI is a set of APIs. These APIs and the Java-docs are contained in the api jar file of the ear file.

You must use these APIs for your customization purposes; do not modify or bypass these APIs unless explicitly instructed in this document.

The reference UI demonstrates how CBA functions. The UI is customizable. For example, you can customize the UI of reporting or even hierarchy. However, because of the complexity of the hierarchy management UI, **we strongly recommend that you try and keep your UI as close as possible to the reference hierarchy UI to reduce your workload**.

The functionalities exposed by the APIs actually exceed the ones demonstrated through the reference UI. Please consult the API Java-docs and other sections of this guide for details. You can customize the CBA application to take advantage of these functionalities.

When you have to change existing CBA files, such as a JSP or a Velocity template, you can either work on an existing file or copy it and then work on the copy. The second method may be more time consuming but will save you more time for migration. Either way, you should keep the history of customization changes in a source control system.

Ear-Repackaging

Whenever you want to modify a JSP, add a new action class or an EJB, you need to re-package the ear files. In general, we expect you are going to re-package the online ear but not the Command Center ear.

When re-package the ear, make sure you don’t remove existing components and only modify the components which are recommended as modifiable in this guide, such as JSP pages, CSS file, app-resources.jar, etc.

UI Customization

UI customization may be as simple as changing the look-and-feel or as complex as adding your own struts action classes.

Customizing the Existing Look-and-Feel

The CBA UI is based on Tiles definition and the UI properties like color, font size, etc. are all controlled through a style sheet.

You can modify the CSS definitions in this file to change the look and feel of the CBA. All CSS definitions are in this file: tam-tbm.ear/war-tbm-b2b.war/tam/_assets/css/screen.css.

You can modify the Tiles definitions file to use your own Tiles. All the Tiles definitions are in this file: tam-tbm.ear/war-tbm-b2b.war/WEB-INF/tiles-def-tam.xml.

The hierarchy UI-related JSP pages are in tam-tbm.ear/war-tbm-b2b.war/tam/hierarchy and tam-tbm.ear/war-tbm-b2b.war/tam/reporting and if needed, you can customize these JSP pages.

In special cases for the UI when you use the CBA presentment engine to generate a report or a search page, the result of the query is not presented by JSP, instead, a set of Velocity templates are used. These templates are defined in <EDX_HOME>/templates/common/lib and reporting directories. You should not touch the VM files in the lib directories. For the files under reporting, you can customize them if necessary. However, we find that most of the time you can customize reports using report xml files without touching the VM files.

Custom JSP Pages and Action Classes

It is possible that you may need to add your own UI components such as JSP pages, JavaScripts etc. If possible, we recommend that you add these UI components into a separate directory under ear-tam-tbm.ear/war-tbm-b2b.war/tam for easier migration. Because the “tam” directory is defined as a Struts module, you need to prefix your URI with “tam”. For example, http://localhost:7001/tbmb/tam/my-custom/my.jsp

After you create your own action class, you must modify the struts-config-tam.xml file to register it.

Velocity Templates

The CBA reporting UI is based on Velocity templates, an open source project. The product offers a set of out-of-the-box templates to implement common UI features such as paging, sorting, charting, print-friendly, and download. To get more details, refer to the CBA *“Reporting Developer’s Guide.”*

You can customize these out-of-the-box templates either by modifying them directly or by copying and then modifying. If you do copy/modify, you need to configure report xmls to pick up your new templates.

Reports

All the report XML files defined in <EDX_HOME>/config/rpt are for the out-of-the-box CBA UI. You can add your own report xml files by following the instructions in the CBA *“Reporting Developer’s Guide.”* Your reports can use either the default Velocity templates provided with CBA or your own templates.

Localization

You can change the resource bundle files to suit your own application localization needs. The resource bundles are all packaged inside a jar file, tam-tbm.ear/xma/app-resources.jar.

Change the URL Prefix

CBA uses “tbmb” as its URL prefix. However, you can change this to fit your deployment environment. The prefix “tbmb” is defined in the application.xml of the ear-tam-tbm.ear/META-INF directory. All URLs from CBA use a relative URL (and no hard-coded references to “tbmb”). This ensures that after you change the URL prefix, the application can still work.

As you already know, the CBA related action classes and other Struts stuff are defined as a Struts module, “tam”. Because of this, access to the Struts action classes should be prefixd with tam, such as “tbmb/tam/report.do”. Though you could change the “tbmb” to something else as stated above, we don’t recommend that you change “tam”.

Spring (XMA) Configuration Files

CBA uses Spring to manage Bean creation and transactions. Even the configuration of Hibernate is through Spring. These files are also called XMA configuration files in CBA terms and exist in the <EDX_HOME>/xma directory. They are the core configuration files of CBA and you should not touch them unless instructed in this document.

Possible reasons to customize these files include:

- Enable Hibernate “show_sql”. See the debugging guidelines for detail.
- Extend hierarchy management, such as adding a new link target type, re-implementing a hierarchy search interface such as “IAssignedObjectProvider”, inserting a new loader into the OLTPPorductionLoader job, configuring a new event handler to handle hierarchy events, or configuring the hierarchy UI behavior. See the CBA *“Hierarchy Developer’s Guide”* for details.
- Configure the Batch Report job, such as whether to send emails. This is documented in the CBA *“Installation Guide.”*

OLTP Database

OLTP is the CBA transactional database. We expect access to product tables to go through CBA APIs. You should not change the existing product schema. However, it is ok to add your own customization tables.

OLAP Database

OLAP is the CBA data warehouse. It is a non-transaction database used to save billing information and has no APIs to access. Instead they are accessed directly through report XML files. For information on how to use report XMLs to retrieve data from OLAP database, check the report XML files used to generate various billing reports. These files are defined in <EDX_HOME>/config/rpt.

The OLAP database includes three kinds of tables:

- **OLTP-OLAP Synchronization-Related Tables**: Since OLAP is a non-transactional database, it has no (or very limited) UI transactional operations. However, it needs transaction data to report on, for example, hierarchy information. This information is synchronized from OLTP to OLAP at real time. The tables related to this operation are:

  EDX_RPT_ACCOUNT_WSPACE
  EDX_RPT_ACCOUNT_XREF (Not used.)
  EDX_RPT_CC_CHARGE_WSPACE
  EDX_RPT_HIERARCHY_NODE_PERIOD
  EDX_RPT_HIERARCHY_TREE_DIM
  EDX_RPT_HIERARCHY_TYPE_DIM
  EDX_RPT_HIERARCHY_XREF_DIM
  EDX_RPT_USER_HIERARCHY_WSPACE (Not used.)
  EDX_RPT_USER_SERVICE_WSPACE (Not used.)

  These tables are expected to be used as-is and you should not try to customize them. The operations we expect you do on these tables should be “read-only”.

- **Dimensional Tables**: Except the ones mentioned above, the remaining dimensional tables are used to save dimensional data such as accounts, services, dates, periods, etc. Also most all the dimensional tables have some flexible fields which are for customization. You should try to use the flexible fields to hold your custom information instead of adding your own columns. You can also create new dimensional tables.

- **Fact Tables**: Tact tables are used to hold the fact information such as call details or summaries. You can add new columns to the fact tables if necessary or add your own fact tables.

You should never make any changes that could break the backward compatibility of the DB schema, such as changing the column type or renaming a column or a table.

4 CBA Debugging Guidelines

Overview

CBA produces various log information for you to use to debug problems. For historical reasons, three logging mechanisms exist in the product:

- **Log4j** – Log4j is the main logging mechanism. Because there are two ear applications, each one of them needs to use different log4j files to avoid stepping on each other. See the CBA "*Installation Guide"* for details.
- **DB-logging** - Most Command Center jobs also use DB-logging for job-level information and log4j is still used to log API-level information. The DB-logging writes log information into DB tables and can be viewed from the Command Center.
- **Java-option-logging** - The logging is controlled by pass-in a JVM "-D" option. This is usually used to log debug-level information and mostly for development purpose.

In addition, in the majority of use cases, CBA prints out the exception stack trace to the console or as part of the JSP error output page when an exception occurs.

Viewing log4j Logs

By default, the online application logs information logged through log4j into cba.log and Command Center logs into estatement.log.

You can change the log level of log4j by configuring the two log4j.xml files, one for the online application and another for the Command Center. See the log4j documentation for details.

Viewing Command Center Logs

Command Center jobs use a combination of DB-logging and log4j to log information.

When there is a problem with a Command Center job, you can view the db-logging for log4j logs.

To view DB logging, log into Command Center, click on "Reporting" -> "View Logs" and then fill in the search criteria.

If DB logging does not provide enough information, you can go to estatement.log which contains log4j information.

Displaying SQLs

One of the most useful debug features is to display the actual SQLs being issued to the database.

To view the Hibernate SQLs, you need to search through all the .xma.xml files in the <EDX_HOME>/xml directory and change this property to “true:”

hibernate.show_sql

However, this property allows you to view the SQLs but not the SQL binding values. To get the binding values, you need to go to log4j.xml and change the log level for these two loggers to “debug”:

```
<logger name="net.sf.hibernate.SQL" additivity="false">
        <level value="error"/>
         <appender-ref ref="cba-log"/>
</logger>

<logger name="net.sf.hibernate.type" additivity="false">
        <level value="error"/>
         <appender-ref ref="cba-log"/>
</logger>
```

These configurations apply to Hibernate-based DB access. One exception is reporting-related SQLs, which are issued without using Hibernate. To view the report SQLs and their binding values, add a Java “-D” option:

```
java –Ddatasource.debug=true
```

5 CBA Migration Guidelines

Overview

CBA provides a development platform to develop custom features. To ensure that the application can be migrated from an old version to new version, you must follow these guidelines.

Installing a New Version of CBA

You should not install a new CBA release on top of the old CBA install directory. Either install the new CBA into a new directory, which requires you re-configure startWebLogic scripts to point to the new location, or move the old CBA to another directory and install the new CBA in the old directory.

The reason we don't recommend installing a new CBA on top of the old one is that some old XMA configuration files used in old version of CBA may not be overwritten in the new version and CBA may fail to load these old configuration files.

Database Migration

CBA provides migration scripts to upgrade from some old versions of CBA. It is possible that you may have a very old version of CBA and there is no direct upgrade from that version to latest; it may be necessary to upgrade to an intermediate version and then to the latest.

The migration scripts guarantee that all your old data will be migrated to the latest CBA db schema. Because of this, it is critical to follow the database customization guidelines listed above.

Code/EAR Migration

It is almost certain that you would customize some of our existing JSP pages and add your own action classes or JSP pages which may call CBA APIs. You must migrate these custom codes as well.

To ensure smooth code migration, CBA makes APIs as backward compatible as possible.

There are two kinds of APIs:

- The majority of the APIs are the ones for which CBA offers implementations and the custom codes can call them. No customizations are expected on these APIs and they should be used as it is, such as IReport and IHierarchy. If there are any changes to these APIs, CBA usually deprecate the old APIs and adds the new ones. The deprecated APIs should still work for old functionalities but may not work for newly added functionalities. This ensures that your custom codes can still function in the olds ways but offers you a path to migrate to new APIs.
- There is a set of other APIs for which CBA only offers default implementations but the custom codes could offer their own implementations, for example, IAssignedObjectProvider, IUnassignedObjectProvider, and IReportObjectResultSet. These interfaces are also called callback or plug-in interfaces. Because of this, any change to these interfaces could affect your implementations. In this case, we will release note the changes to these interfaces.

Follow these steps to migrate your codes:

1. Get the latest CBA libraries.
2. Re-compile your custom Java classes. If an API has been deprecated, you will see a deprecation warning and you need to check the API document to upgrade to the non-deprecated APIs. If a callback interface has been changed and backward compatibility being broken, you will see a compilation error. In this case, consult the API document to use the new API.
3. For your custom JSP pages, you may want to use JSP pre-compiler which comes from Application Server vendors to check the code. If this is not possible, you can use IDE to go through your JSPs, which may point to you where the deprecated APIs are used and where a compilation error may occur.
4. Repackage your custom code against the new CBA product ears to create your custom deployment ears. These include Java code, JSP code, web resources like HTML pages, JavaScripts etc. See the customization guidelines for how to package properly.
5. Redeploy the ear and re-test. Some times we find that an ear may not be re-deployed without re-starting the application server. This is usually an application server issue and can be solved by upgrading to the latest patch of the application server. If this doesn't help, restart your application server if re-deployment fails.

Command Center Job Migration

You must recreate and reconfigure all necessary CBA jobs in the Command Center.

New features often require new configuration parameters; because these new parameters may not even have default values, it is not possible to migrate a CBA job to new configuration.

Configuration Migration

In addition to database and ear migration, you must migrate the configuration files in the CBA_HOME and EDX_HOME directories as well.

XMA Configuration Files

To migrate XMA configuration files:

1. Record and copy the list of XMA configuration files you have customized.
2. Install a new CBA as stated above.
3. Although it is possible that your customized XMA files may work in the new CBA, it is strongly recommended that you compare the custom XMA files with the ones from the new CBA and manually merge your changes if you find that new content has been added to the files by the new CBA. A source control system or other tool may help you compare.

Report XML Files

Backward capabilities of report XML files are guaranteed unless stated otherwise. This means your old report XMLs can work with new CBA without any changes. You can copy your own report XML files back to the EDX_HOME/config/rpt directory. However, in order to take advantage of new report functionality, you need to modify your report xml files.

Report Template Files

The Velocity template files are like the JSP pages; the old templates are expected to work with the new CBA without any changes.

As stated above, in general, we don't expect you customize the out-of-the-box templates. If you do, you may have to manually merge these template files to take advantage of new CBA report features or bug fixes.

To migrate report template files:

1. Make copies of the templates you have customized.
2. Install a new CBA and its templates as stated above.
3. Compare the customized template with the new template and merge the changes you've made into the new template.

Application Server Configuration

You may need to change the application server configuration, such as the startWebLogic script or the JDBC connection pool configurations. Consult the CBA "*Installation Guide*" to check your current configuration against the new configuration.

Test, Test, and Test

Be sure to perform a full regression test on the migrated application before going live.