Sterling Selling and Fulfillment Foundation

Extending the Database

Version 9.1

Sterling Selling and Fulfillment Foundation

Extending the Database

Version 9.1

Note
Before using this information and the product it supports, read the information in "Notices" on page 49.

Contents

Chapter 1. Checklist for Customization Projects 1

Customization Projects 1
Prepare Your Development Environment 1
Plan Your Customizations 1
Extend the Database 1
Make Other Changes to APIs 2
Customize the UI 2
Extend Transactions 2
Build and Deploy your Customizations or Extensions 3

Chapter 2. Guidelines for Extending Databases 5

Guidelines for Extending Databases 5
About Extending the Database 5
What is the SQLProxy Tool? 6
Setting Up the SQLProxy Tool 6
Guidelines for Adding Columns to a Standard Table 7
Guidelines for Adding Support for Data Compression for Columns 8
Guidelines for Adding Non-Unique Indices to a Standard Table. 9
Guidelines for Adding Foreign Key Elements to a Standard Table. 9
Guidelines for Adding Text Search Index Elements to a Standard Table 9
Coding Guidelines to Avoid Deadlocks 9

Chapter 3. Extending Database Tables 11

Adding a Column to a Standard Table 11
Features Requiring Multiple Extensions When Adding a Column to a Standard Table 13
Increasing the Size of a Standard Column 14
Extending a Catalog Search 15
Predefining and Grouping a Set of Columns for a Query 18
Adding Unique Tag Identifiers and Descriptors to a Standard Table 19
Adding Non-Unique Indices to a Standard Table . . 20
Adding Foreign Key Elements to a Standard Table 21
Adding Text Search Indices to a Standard Table . . 23
Enabling Case Insensitive Searches 24
Modifying an Entity XML 25
Creating Custom and Hang-off Tables 25
Steps to Create a Custom Table 26
Steps to Create a Hang-off Table 31
Purging Data from Hang-off Tables 35

Chapter 4. Extending the Data Types Files 37

Extending the Data Types Files 37
Extending the Data Type Map File. 37
Extending the Data Type File 37

Chapter 5. Create a Custom View . . . 39

Create a Custom View to Join Multiple Tables . . . 39

Chapter 6. Generating Audit References for Entities 41

About Extending Audit References 41

Chapter 7. Extending API Templates . . 43

About Extending API Templates 43
Including Extended Attributes in the API Template 43
Including Custom and Hang-Off Entities in the API Template 44
Configuring Services for Custom and Hang-off APIs 45

Notices 49

Chapter 1. Checklist for Customization Projects

Customization Projects

Projects to customize or extend Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales vary with the type of changes that are needed. However, most projects involve an interconnected series of changes that are best carried out in a particular order. The checklist identifies the most common order of customization tasks and indicates which guide in the documentation set provides details about each stage.

Prepare Your Development Environment

Set up a development environment that mirrors your production environment, including whether you deploy your application on a WebLogic, WebSphere®, or JBoss application server. Doing so ensures that you can test your extensions in a real-time environment.

You install and deploy your application in your development environment following the same steps that you used to install and deploy it in your production environment. Refer to your system requirements and installation documentation for details.

You have an option to customize your application with Microsoft COM+. Using Microsoft COM+ has advantages such as increased security, better performance, increased manageability of server applications, and support for clients of mixed environments. If this is your choice, see the *Customization Basics Guide* about additional installation instructions.

Plan Your Customizations

Are you adding a new menu entry? Or customizing the sign-in screen or logo? Or customizing views or wizards? Or creating new themes or new screens? Each type of customization varies in scope and complexity.

For background, see the *Customization Basics Guide*, which summarizes the types of changes that you can make and provides important guidelines about file names, keywords, and other general conventions.

Extend the Database

For many customization projects, the first task is to extend the database so that it supports the other UI or API changes that you make later. For instructions, see the *Extending the Database Guide*, which includes information about the following topics:

- Important guidelines about what you can and cannot change in the database.
- Information about modifying APIs. If you modify database tables so that any APIs are impacted, you must extend the templates of those APIs or you cannot store or retrieve data from the database. This step is required if table modifications impact an API.
- How to generate audit references so that you improve record management by tracking records at the entity level. This step is optional.

Make Other Changes to APIs

Your application can call or invoke standard APIs or custom APIs. For background about APIs and the services architecture of service types, behavior, and security, see the *Customizing APIs Guide*. This guide includes information about the following types of changes:

- Invoke standard APIs for displaying data in the UI and for saving changes made in the UI to the database.
- Invoke customized APIs for executing your custom logic in the extended service definitions and pipeline configurations.
- APIs use input and output XML to store and retrieve data from the database. If you don't extend these API input and output XML files, you may not get the results you want in the UI when your business logic is executing.
- Every API input and output XML file has a DTD and XSD associated to it. Whenever you modify input and output XML, you must generate the corresponding DTD and XSD to ensure data integrity. If you don't generate the DTD and XSD for extended XMLs, you may get inconsistent data.

Customize the UI

IBM® applications support several UI frameworks. Depending on your application and the customizations you want to make, you may work in only one or in several of these frameworks. Each framework has its own process for customizing components such as menu items, logos, themes, and so on.

Depending on the framework you want, consult one of the following guides:

- *Customizing the Console JSP Interface Guide*
- *Customizing the Swing Interface Guide*
- *Customizing User Interfaces for Mobile Devices Guide*
- *Customizing the Rich Client Platform Guide* and *Using the RCP Extensibility Tool Guide*
- *Customizing the Web UI Framework Guide*

Depending on the framework you want, consult one of the following guides:

- *Customizing the Console JSP Interface Guide*
- *Customizing the Swing Interface Guide*
- *Customizing User Interfaces for Mobile Devices Guide*
- *Customizing the Rich Client Platform Guide* and *Using the RCP Extensibility Tool Guide*
- *Customizing the Web UI Framework Guide*

Extend Transactions

You can extend and enhance the standard functionality of your application by extending the Condition Builder and by integrating with external systems. For background about transaction types, security, dynamic variables, and extending the Condition Builder, see the *Extending Transactions Guide* and *Extending the Condition Builder Guide*. These guides includes information about the following types of changes:

- Extend the Condition Builder to define complex and dynamic conditions for executing your custom business logic and using a static set of attributes.

- Define variables to dynamically configure properties belonging to actions, agents, and services configurations.
- Set up transactional data security for controlling who has access to what data, how much they can see, and what they can do with it.
- Create custom time-triggered transactions. You can invoke and schedule custom time-triggered transactions in much the same manner as you invoke and schedule the time-triggered transactions supplied by your application.
- Coordinate your custom, time-triggered transactions with external transactions and run them either by raising an event, calling a user exit, or invoking a custom API or service.

Build and Deploy your Customizations or Extensions

After performing the customizations that you want, you must build and deploy your customizations or extensions.

1. Build and deploy your customizations or extensions in the test environment so you can verify them.
2. When you are ready, repeat the same process to build and deploy your customizations and extensions in your production environment.

For instructions about this process, see the *Customization Basics Guide* which includes information about the following topics:

- Building and deploying standard resources, database extensions, and other extensions (such as templates, user exits, and Java™ interfaces).
- Building and deploying enterprise-level extensions.

Chapter 2. Guidelines for Extending Databases

Guidelines for Extending Databases

Modifying sequences is not supported. Editing the sequences.xml file in the entity repository will have no effect. Furthermore,IBM recommends against modifying sequences via the database tools. Under most circumstances it will not be necessary and will have adverse effects.

About Extending the Database

Certain aspects of theSterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales database cannot be modified. If you try to make these modifications, your data is not harmed, but your attempted changes are not incorporated into the database. The application does not permit modification of the following:

- Existing columns of tables
- Primary keys of tables
- Unique keys of tables
- Views

When planning extensions to the database, consider the implications of your changes and how they may impact other areas.

Note: If you modify a table and your deployment uses the Sterling Business Centercomponent, the view associated with the table must also be modified.

Entity Relationship Diagrams

To learn more about the Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales database, see the entity relationship diagrams (ERDs) using the *INSTALL_DIR*/xapidocs/erd/html/erd.html file. These ERDs provide the following information:

- Indicate which tables can be extended by adding columns.
- Indicate which tables can have hang-off relationship.
- Relationships between tables (to help you understand the relationship between logical entities such as orders, shipments, and payments).
- Indices details. Each table is indexed by a primary key. Most tables also have a unique index that is constituted of the columns that make the logical unique key. In addition, some tables have alternate indices to support queries.
- Views that indicate how several tables interact.

Entity Database XML Files

The standard tables that are shipped with the application database are defined in a set of entity XML files, also known as database definition XML files. Each entity XML file may contain several table definitions. To learn more about these tables, see the files in the *INSTALL_DIR*/repository/entity directory. Within these entity XML files, an entity represents a table and an attribute represents a column.

What is the SQLProxy Tool?

SQLProxy captures SQL statements going through a JDBC driver to the database and generates JDBC traces. SQLProxy captures all SQL statements, regardless of the type of database being used. Another tool, the SQL Proxy Analyzer, produces a report based on the SQLProxy trace records. Using the SQLProxy tools can help you to analyze database response times for queries, which can help your capacity planning and maximize efficient usage of your database servers. Only use these tools when directed to do so byIBM Customer Support.

Setting Up the SQLProxy Tool

About this task

To set up the SQLProxy tool, you must add properties to the customer_overrides.properties file prior to starting the application server or noapp server. These will override settings for these properties in jdbc.properties.

Note: The SQLProxy properties in the yfs.properties file (such as yfs.enable.proxy.sql.logging, yfs.enable.source.logging, and yfs.proxy.log.dir) have been deprecated. All users, regardless of whether you are using a third party application server (WebSphere, JBoss, WebLogic) or the noapp server, must configure the tool by adding the following JDBC properties to the customer_overrides.properties file.

Procedure

1. In the *install_dir*\properties\ folder, locate the customer_overrides.properties file or create it, if it does not exist.
2. Set the following properties in customer_overrides.properties:

   `jdbcService.proxyLoggingEnabled=Y`

   (Determines if logging is needed.)

   `jdbcService.proxySourceLogging=Y`

   (Provides additional caller information.)

   `jdbcService.proxyLogDir=<INSTALL_DIR>\logs`

   (Directory where the trace files will be generated.)

Starting and Using the SQLProxy Tool

About this task

Logs are generated for all database operations when the log level is set to TIMER or higher. To begin tracing, execute the API or agent after setting the log level. The trace files are generated in the location specified in the proxyLogDir property. The following details are included in the files:

- *.log file gives component name, start and end system times, operation execution times (individual and cumulative), operation name, SQL statement, java class name
- *.tail file helps in understanding the long running queries (used mostly in performance testing)

Stopping the SQLProxy

About this task

To stop logging, remove the TIMER trace. To completely disable the proxy, set the enabled property to `N`. See *Stopping a Component Trace* in the Sterling Business

CenterSterling Selling and Fulfillment FoundationSterling Field Sales*System Management Guide.*

Analyzing the Results

You can export the contents of a file to a .csv file and open it in XSL. There is also a support tool called the SQLProxy Analyzer which analyzes and summarizes JDBC trace records. This tool is available as part of MTCP in Eclipse.

CAUTION:
ContactSterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales Customer Support when using this tool.

About the SQLProxy Analyzer

The SQLProxy Analyzer is a tool used to analyze the data in the SQLProxy logs. The Analyzer has the following capabilities:

- Can process multiple log files in a directory or a single log file
- For each log file (which represents a database connection), the Analyzer produces a summary of the unit of work (UOW). (There might be multiple UOWs in a connection.)
- Can calculate the number of executions and total cost of the SQL
- Response Time Buckets, which are useful if you have a many SQL instances and a high average SQL time. The response time bucket will show you which bucket the SQL response time falls into.
- Provides execution plans, which are automatically generated by the Analyzer. You can specify any Oracle database (it does not necessarily have to be the one from which the SQL was captured). This enables you to point the analyzer to a very large database that has up-to-date Oracle statistics.
- For each unit of work, the Analyzer provides:
  - A summary of all the SQL (SQLs are grouped together to get a small summary from a very large file)
  - The LOCK HOLDING time (therefore, you can see if the system grabs locks that could impact other workloads)
  - Oracle EXPLAIN plans, which assist you in assessing whether certain queries will potentially get slower as the database grows larger

Guidelines for Adding Columns to a Standard Table

When extending the columns of standard tables, keep the following considerations in mind:

- You can only add columns to tables as specified in the ERDs.
- You cannot remove or modify any columns.
- You can add columns either before or after installing the application.
- For all columns added to an application database table, you must provide a default value that is relevant to the database framework.
- You cannot use nullable columns for the following fields:
  - Primary Key Attributes
  - Entity Relationships

  Hence, in the entity XML, `Nullable="true"` is allowed for all columns except the ones noted above.
- You cannot add columns with a data type of `Long`.

- When using application components (such as events and user exits) that read in a map or publish a map (such as the GetOrderNoUE user exit), extended fields in the maps are prefixed with Extn_.

Note: In Oracle database, data type CLOB is generated instead of LONG for new installations. However, for upgrades, existing LONG columns will remain unchanged.

Note: In DB2® database, the Date data type is generated as TIMESTAMP by theSterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales framework.

Guidelines for Adding Support for Data Compression for Columns

You can enable data compression for new and existing columns of Standard, Custom, or Hang-off tables. The data compression allows a reduction in size for tables when being implemented.

- For newly added columns, you can enable data compression by adding both `CompressionSupported="true"` and `UseCompression="true"` in the entity XML.
- For existing columns in a standard table, you can enable data compression by overriding the value of the UseCompression attribute.

  Note: You may only enable data compression for existing columns that have the CompressionSupported attribute set to `True`.

When adding support for data compression to the columns of standard tables, keep the following considerations in mind:

- Data Compression should only be used for columns containing a large amount of text data that is typically not queried directly, such as audit records, error stack traces, and XML template data. The nature of these records can consume a significant amount of space within the database.
- Columns using CompressionSupported="true" do not support query operations from list APIs. This can be overridden by setting the QueryAllowed attribute to True in the entity XML.
- By default, columns using `CompressionSupported="true"` cannot be used as part of an index or unique constraint. This can be overridden by setting the AllowCompressedColumns attribute to True on the index element in the entity XML.
- Primary Key columns cannot be marked as compressible columns.
- Data compression should be enabled only if the maximum size of the column is >= 500 bytes. Any column marked for data compression with less than 500 bytes results in warnings.
- By default, the data compression is done using the GZip algorithm. You can override this default data compression logic by setting the yfs.db.compression.class=*class name* property in the customer_overrides.properties file. In the *class name*, specify the name of your custom class which contains the data compression logic.

Data Compression Logic

The data for compressible columns is compressed using the GZip algorithm. You can override this default data compression logic and provide your custom compression logic by implementing the SCICustomDataCompressor interface.

You must enter the name of the custom class that implements the SCICustomDataCompressor interface in the yfs.db.compression.class=*class name* property in the customer_overrides.properties file.

Guidelines for Adding Non-Unique Indices to a Standard Table

When adding non-unique indices, use a naming convention that differs from theSterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales convention, which is *tablename*_i<1+n>. Using your own naming convention prevents your indices from accidentally being dropped during upgrades. The following considerations are also recommended:

- Adding a prefix that doesn't start with Y.
- Prefix your non-unique indices with EXTN_ for easier identification.
- Unique indices are not allowed for tables.
- Column names for indices must be valid.
- Index names should not exceed 18 characters.

Guidelines for Adding Foreign Key Elements to a Standard Table

Currently foreign key relationships in extended columns of tables are restricted to only the YFS_PERSON_INFO table. When exposing foreign key elements, the following validations are performed:

- The parent table name must be YFS_PERSON_INFO.
- The parent column name must be a primary key of the YFS_PERSON_INFO table.

Guidelines for Adding Text Search Index Elements to a Standard Table

When adding text search index elements to standard tables, keep the following considerations in mind:

- You can add the text search element only to non-transactional tables.
- The searches performed for the text search indices are case insensitive.
- You can define more than one text search indices per entity.
- You cannot define multiple columns on a text index.
- IBM supports CTXCAT and CONTEXT text search index types on Oracle.
- Defining both the extended entity and the text search index in the same extension XML file is recommended.

Coding Guidelines to Avoid Deadlocks

Deadlock refers to a specific condition in a database, when two processes are waiting for the other process to release a resource. For example, when one client application holds a lock on a table and attempts to obtain the lock on a second table that is held by another client application, this may lead to a deadlock if the other application attempts to obtain the lock that is held by the first application.

To circumvent the deadlock problem, it is recommended that you need to sort the information to be accessed in a certain order before grabbing locks. This is applicable to situations where you need to grab multiple inventory item locks within a single transaction boundary. However, you do not need to sort if you call the APIs to process single items per transaction commit.

Reading Uncommitted Data in DB2 Database

In DB2, when you select a record from a table, a read lock is obtained on the record. If the record being selected has been updated but not committed, the thread waits until it commits the changes. Alternatively you could read the record with Uncommitted Read (UR) in which case the latest value that has been updated is provided to the user.

You can read uncommitted data from any list API by enabling the ReadUncommitted attribute to Y in its input XML. To achieve this, you must customize the individual JSP and pass the ReadUncommitted attribute as a hidden attribute. For example:

```
<input type="hidden" name="xml:/Order/@ReadUnCommitted" value="Y"/>
```

As a result, the locking scenario is circumvented in the DB2 database. Locking is the default in DB2.

It is not mandatory to pass this flag. However, if you set this flag to Y, the system is forced to read uncommitted data. For example, a transaction, T1, updates Table TAB-1 but the transaction's data is not committed. If the ReadUncommitted flag is set to Y, other transactions can read the uncommitted data in Table TAB-1.

Before setting this flag, evaluate concurrent transactions to determine whether a situation exists in which a deadlock is occurring. If no such situation occurs, the flag should remain at its default setting.

This behavior is different from Oracle, hence if you are writing custom code on DB2 you should understand this behavior to avoid lock escalations.

Chapter 3. Extending Database Tables

Adding a Column to a Standard Table

About this task

You add columns to tables by modifying the entity database extension XML files and then rebuilding the application database and JAR files. After the application has been rebuilt, the APIs recognize these added columns and use them when storing and retrieving data.

To add a column to a standard table:

Procedure

1. Copy the *install_dir*/repository/entity/extensions/Extensions.xml.sample file as *install_dir*/extensions/global/entities/*your_filename*.xml file OR modify your existing extension XML file.
2. Edit the *your_filename*.xml file to add a new entity tag as shown in the following example for each table you want to extend. If the tag already exists, use the existing one. For a description of the XML attributes, see the table that follows the example.

```
<!-- element exposed to create a column -->
<DBSchema>
  <Entities>
    <Entity TableName="REQUIRED">
    <Attributes>
      <Attribute ColumnName="REQUIRED" DataType="" DecimalDigits=""
       CompressionSupported="false"
       UseCompression="false" QueryAllowed="false" DefaultValue="" Description=""
       Nullable="false" Size="1" Type="REQUIRED" XMLName="" XMLGroup=""
       SqlServerDataType="" />
    </Attributes>
    </Entity>
  </Entities>
</DBSchema>
```

Attribute	Description
ColumnName	Required. Name of the column added to this table. The ColumnName must start with EXTN_.
DataType	Optional. Valid values are available in the *install_dir*/repository/datatypes/datatypes.xml file.
DecimalDigits	Optional. Number of digits of precision required after the decimal. Needed only for numeric fields.
CompressionSupported	Optional. Attribute used to indicate whether or not the data compression is supported for this column. Valid values are `True` or `False`. If True, compression support is enabled. **Note:** If the data inserted into a column has both CompressionSupported and UseCompression attributes set to True, then this attribute should not be set back to False. Doing so would result in all compressed data being retrieved without any decompression.

Attribute	Description
UseCompression	Optional. Attribute used to compress data for this column. Valid values are `True` or `False`. If True data is compressed. **Note:** The value of this attribute should be set to True only if the CompressionSupported attribute is set to True.
QueryAllowed	Optional. Attribute used to enable a compressible column to be used for queries in a List API. Valid values are `True` or `False`. If True the column can be used for queries in the List APIs. **Note:** If the value of the CompressionSupported attribute is set to true then the value of this attribute should also be set to True.
DefaultValue	Required. Used as is for the defaults clause in your database.
Description	Optional. Description of column usage.
Nullable	Optional. Attribute used to describe the nullable value of a field. Default is false. `Nullable=true` is allowed for all columns except Primary Key Attributes and Entity Relationships.
Size	Size of the database column.
Type	Required. Data type of the database column. This attribute also determines the type of attribute in the Java classes that are generated and the format of the attribute in the XML. The valid types are CHAR, VARCHAR2, NUMBER, DATE, and TIMESTAMP. If you are using Microsoft SQL Server® and want to specify a data type as TEXT in the database, you also need to use the SqlServerDataType attribute and specify the attribute value as TEXT. **Note:** If DATE is specified, only the calendar date is stored. If TIMESTAMP is specified, the calendar date and time are stored.
XMLName	XML name of the attribute, if it is different from the name of the attribute. Choose a name that does not conflict with the base extension. It is recommended that you use Extn as a prefix. It is also strongly recommended that you use the same convention for arriving at the XMLName as the Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales base product does: Make each letter following the underscore in the column name upper case, and the rest lower case. Then, remove the underscores. Thus, Extn_Item_Id should be: `ExtnItemId`.
XMLGroup	If present, indicates the child tag in which the attribute is present. If the attribute is not present in the XML, use the NOT_SHOWN string. The XMLGroup must be `Extn`. Thus, the data for the extended columns is in a separate element in the API XML output.

Attribute	Description
VirtualDefaultValue	Optional. This attribute is applicable to Null Columns. When a null is returned from the database, it is stored in memory as the virtual default value.
SqlServerDataType	Optional. Pertains only to Microsoft SQL Server databases. If you see a warning about the row size being too long, specify one or more of your larger columns as "TEXT". Columns of type TEXT are not included in the maximum row size calculation for a table.
ForceUpperCase	Optional. If a "case insensitive" search is required for a text column, this field should be set to "True". If set to "True", the system converts the data entered in this field to uppercase.

3. Create a new Attribute tag for each column you want to add to the table.
4. Use the database verification tool dbverify for generating scripts to add the columns to your database.

   Note: On Microsoft SQL Server, the total length of all extended columns should not exceed 900 bytes. If Microsoft SQL Server throws a warning that the row size exceeds the maximum length, change the data type of one or more of your columns to TEXT and then specify TEXT for the SqlServerDataType attribute as described in the table above.
5. Extend the corresponding API templates by following the steps described in "About Extending API Templates."
6. Build and deploy your extensions.

Results

A special case of extending columns for adding unique tag identifiers or descriptors is explained in “Adding Unique Tag Identifiers and Descriptors to a Standard Table” on page 19.

Features Requiring Multiple Extensions When Adding a Column to a Standard Table

In addition to extending standard tables, you can extend the following features using the specified guidelines:

- Classification Inheritance—If you are adding columns to extend attributes in the YFS_ITEM table and you want to make these attributes available for classification inheritance, a duplicate entity tag must be added to your XML for the YFS_CLASS_ITEM_ATTR table. Additionally, Nullable should be set to `true` and DefaultValue should not be passed for these attributes. For more information on defining item attributes at the classification level, refer to the *Catalog Management Configuration Guide*.
- Item Entitlements—If you are adding columns to extend attributes in the YFS_CUSTOMER table and you want to make these attributes available for entitlement rule assignment, a duplicate entity tag must be added to your XML for the YFS_ENTITLE_RULE_ASSIGNMENT table. Additionally, Nullable should be set to `true` and DefaultValue should not be passed for these attributes. For more information on entitlement rule assignment, refer to the *Business Center: Item Administration Guide*.

- Pricelist Assignments—If you are adding columns to extend attributes in the YFS_CUSTOMER table and you want to make these attributes available for pricelist assignment, a duplicate entity tag must be added to your XML for the YFS_PRICELIST_ASSIGNMENT table. Additionally, Nullable should be set to `true` and DefaultValue should not be passed for these attributes. For more information on pricelist assignment, refer to the *Business Center: Pricing Administration Guide*.
- Pricing Rule Assignments—If you are adding columns to extend attributes in the YFS_CUSTOMER table and you want to make these attributes available for pricing rule assignment, a duplicate entity tag must be added to your XML for the YFS_PRICING_RULE_ASSIGNMENT table. Additionally, Nullable should be set to `true` and DefaultValue should not be passed for these attributes. For more information on pricing rule assignment, refer to the *Business Center: Pricing Administration Guide*.

Increasing the Size of a Standard Column

About this task

You can increase the size of the table columns by extending the yfsdatatype XML file and rebuilding the database and JAR files. After the application is rebuilt, the APIs recognize these resized columns and use them when storing and retrieving data. This can only be done for columns of datatype VARCHAR2, and the size of the columns can only be increased, not decreased.

To increase the size of a standard column in a table:

Procedure

1. Create a new *install_dir*/extensions/global/etc/datatypes.xml file, if it does not already exist.

   If the /global/etc/ directory structure does not exist, create the required directory structure.
2. Copy the existing column definition from the *install_dir*/repository/datatypes/datatypes.xml file to the newly created *install_dir*/extensions/global/etc/datatypes.xml file and then apply the required changes.
3. Extend the datatype by increasing the size attribute to the desired size as shown in the following example. Only the desired changes need to be listed in this file. Do not mention datatypes that do not need to be changed.

   ```
   <DataTypes>
   <DataType Name="TagNumber" Size="140"/>
   </DataTypes>
   ```
4. Manually alter the columns in the database. You can also use the database verification tool dbverify for generating scripts to add columns to your database.

   Note: On Microsoft SQL Server, the total length of all extended columns should not exceed 900 bytes. If the Microsoft SQL Server throws a warning that the row size exceeds the maximum length, change the datatype of one or more of your columns to TEXT. Specify TEXT for the SqlServerDataType attribute as described in the Attributes table in "Adding a Column to a Standard Table."
5. Build and deploy your database extensions.

Extending a Catalog Search

About this task

You can extend a catalog search to provide broader search capabilities by updating information in the catalog search index file. To update the index file, modify the corresponding extended XML configuration file. The following components of catalog search are extensible:

- Search system configurations
- Locales and the corresponding analyzer
- Query parser
- Attributes

To extend a catalog search:

Procedure

1. Copy the *install_dir*/repository/xapi/template/merged/resource/extn/ExtnCatalogSearchConfigProperties.xml.sample file as *install_dir*/extensions/global/template/resource/extn/ExtnCatalogSearchConfigProperties.xml file OR modify your existing extension XML file.

   The following example shows a sample ExtnCatalogSearchConfigProperties.xml file.

```
<SearchConfigurations>
  <SearchSystemConfigurations>
    <MergeFactor Value="2"/>
      <MaxMergeDocs Value="2147483647"/>
   </SearchSystemConfigurations>
    <IndexSets>
    <IndexSet Name="CatalogIndex">
      <Locales>
       <Locale LocaleCode="en_US"
        SynonymFile="/properties/EnglishSynonym.properties"/>
        <Locale LocaleCode="fr_FR"
          QueryParserClass="package.FrenchQueryParser.class"
          AnalyzerClass="package.FrenchAnalyzer.class"
          SynonymFile="/properties/FrenchSynonym.properties"/>
       </Locales>
       <Entities>
          <Entity Name="Item">
             <Attributes>
               <Attribute XMLName="MyExtendedDescription"
                 DefaultWeight="1.0" Index="ANALYZED"
                 Store="Y" UseSynonyms="N" Searchable="Y" Sortable="N"/>
               <Attribute IndexFieldName="CustomerItemDescription"
                 DefaultWeight="1.0" Index="ANALYZED" Store="Y" UseSynonyms="N"
                 Searchable="Y" Sortable="N" GetExternalValue="Y"/>
            </Attributes>
             <Entity Name="CategoryItem" RelationshipName="Category_ItemList">
             <Entity Name="Category" RelationshipName="Category">
            <Attributes>
              <Attribute XMLName="MyCategoryExtendedDescription"
                DefaultWeight="1.0" Index="ANALYZED" Store="Y"
                UseSynonyms="N" Searchable="Y" Sortable="N"/>
            </Attributes>
           </Entity>
           </Entity>
           <Entity Name="Asset" RelationshipName="AssetList">
            <Attributes>
              <Attribute Type="MyManual" DefaultWeight="1.0" Index="ANALYZED"
               Store="N" UseSynonyms="N" Searchable="Y" Sortable="N"/>
             </Attributes>
```

```
            </Entity>
           </Entity>
         </Entities>
       </IndexSet>
  </IndexSets>
 </SearchConfigurations>
```

2. To modify system configuration information for the index file, modify the elements in the SearchSystemConfigurations section of ExtnCatalogSearchConfigProperties.xml. The following table describes the SearchSystemConfiguration elements in ExtnCatalogSearchConfigProperties.xml.

Element	Description
MergeFactor Value	Specifies the number of documents that the Apache Lucene open source search engine stores in memory before writing them to disk as a single segment. For more information about setting this value, see: http://lucene.apache.org
MaxMergeDocs Value	Specifies the number of documents that can be contained in one segment. For more information about setting this value, see: http://lucene.apache.org

3. To add or modify locale information for the index file, add or modify the elements in the Locales section of ExtnCatalogSearchConfigProperties.xml. The preceding example shows a sample ExtnCatalogSearchConfigProperties.xml file in which a synonym file has been added to the US-English locale, and a French locale has also been added. The following table describes the Locale elements in ExtnCatalogSearchConfigProperties.xml.

Element	Description
LocaleCode	Specifies the locale code for the index file. If you are adding a locale, ensure that you specify a corresponding analyzer. US-English is the default locale.
QueryParserClass	Specifies the path to the query parser. The application provides the Lucene query parser by default.
AnalyzerClass	Specifies the path to the analyzer. The application provides corresponding analyzers for the default locale.
SynonymFile	Specifies the path to the synonym file for the corresponding locale. Use the synonym file to configure related terms for keyword searches.

4. To add item information for the index file, add the attributes in the Item section of ExtnCatalogSearchConfigProperties.xml. The preceding example shows a sample ExtnCatalogSearchConfigProperties.xml file in which the MyExtendedDescription attribute and the CustomerItemDescription attribute have been added. The following table describes Item attribute elements in ExtnCatalogSearchConfigProperties.xml.

   Note: You can add attributes to the extended XML configuration file for the catalog search index, but you cannot modify attributes nor add entities. See *install_dir*/repository/xapi/template/merged/resource/CatalogSearchConfigProperties.xml for a list of attributes that are supported by default.
5. To add catalog information pertaining to the index file, add the attributes in the Category section of ExtnCatalogSearchConfigProperties.xml. The preceding example shows a sample ExtnCatalogSearchConfigProperties.xml file in which

the MyCategoryExtendedDescription Category attribute has been added. The following table describes the Category attribute elements ExtnCatalogSearchConfigProperties.xml.

6. To add asset information for the index file, add attributes in the Asset section of ExtnCatalogSearchConfigProperties.xml. The preceding example shows a sample ExtnCatalogSearchConfigProperties.xml file in which the asset with the asset type MyManual added. The following table describes attributes for CatalogSearchConfigProperties.xml.

   Note: The application calls the YCMParseAssetUE user exit to parse the content of the asset for the corresponding asset type.

Attribute	Description
XMLName	For item and category attributes of Item Entity, specifies the XML attribute name for the field.
IndexFieldName	Specifies the field name of the attribute as stored in the index. Values for IndexFieldName must be unique throughout the configuration file. If IndexFieldName is not configured in the xml configuration file, the system derives a value for it based on the formula *Entity Name.XMLName*.
Type	For asset attributes, specifies the asset type in the database.
Default Weight	Specifies the weight given to the term. The default value is 1.
Index	Specifies one of the following options for storing field values: Analyzed - stores the value as searchable segments. For example, if a user searches for Desktop Computers and the index parameter for the field is analyzed, the search results include items with the terms Desktop Computer, Desktop, and Computer. Non_analyzed - stores the value as it appears in the database, requiring an exact match to return the value. In the previous example, if the search for Desktop Computers was performed on a non_analyzed field, the search would return only items with the term Desktop Computer.
Store	Specifies one of the following options for storing field values: Y - stores the value of the attribute in the index so that it may be returned as search output. If the GetExternalValue attribute is set to Y, Store is treated as Y. N - does not store the value and only keeps data in the index for the attribute that is in a proprietary format and used only during the search.
UseSynonyms	Specifies one of the following options for including synonyms in the search: Y - indicates that search queries include synonyms. N - indicates that search queries do not include synonyms.

Attribute	Description
Searchable	Specifies one of the following search options for the field: Y - indicates that the field is included as possible search criteria when creating a search query. N - indicates that the field is not included as possible search criteria when creating a search query. For example, the Searchable parameter for the Is_Superseded field in the default XML file is set to N and the Store parameter is set to Y. Users cannot search on the Is_Superseded field in a query. However, queries of superseded items that are obsolete return the superseding items.
Sortable	Specifies one of the following sort options for the field: Y - indicates that search results are sorted by this field. N - indicates that search results are not sorted by this item.
GetExternalValue	Specifies one of the following options for the field: Y - indicates that the value for this attribute is obtained from an external source through a user exit. N - indicates that the user exit is not called.

Predefining and Grouping a Set of Columns for a Query

You can select and group columns to be used in a query (select/list) search by adding a parameter QueryGroup in the entity definition. Each QueryGroup contains the columns to be used for a query. Each QueryGroup is identified by a unique name under which, the columns must be specified.

A global map QueryGroupMap is provided in the Entity class, which contains a unique query group name and attributes corresponding to the columns of the query group. This map is populated when entity repository is loaded.

Note: The QueryGroup name must be not be blank. When adding a query group name to the map, check whether the name already exists in the map.

Each entity can have multiple such query groups, which are defined under a parameter QueryGroups:

```
<Entity>
  <QueryGroups>
    <QueryGroup Name="PasswordQueryGroup">
       <Column Name="PASSWORD"/>
       <Column Name="IS_PASSWORD_ENCRYPTED"/>
       <Column Name="SALT"/>
     </QueryGroup>
     <QueryGroup Name="BusinessQueryGroup">
       <Column Name="BUSINESS_KEY"/>
       <Column Name="CONTACTADDRESS_KEY"/>
       <ColumnName="BILLINGADDRESS_KEY"/>
      </QueryGroup>
   </QueryGroups>
</Entity>
```

Any number of columns can be specified under the query group. However, columns must not be repeated for a query group.

Note: A column in a QueryGroup can be a part of other QueryGroups. A query group must contain at least one valid column and columns defined for a query group must exist in the entity. You cannot add virtual columns to the query group.

To obtain a list of all columns that are part of a particular QueryGroup, use the method getColumnSetForQueryGroup in the following format:

```
public Set getColumnSetForQueryGroup(String queryGroupName)
```

The following methods also can be used along with specific parameters to get a list of columns in a query group:

- selectWithWhereForQueryGroup
- listWithWhereForQueryGroup

For example,

```
public YFS_User selectWithWhereForQueryGroup(YFCDBContext ctx,
String aWhereClause, String queryGroupName) throws YFCDBException{
Set columns = getColumnSet(queryGroupName);
return selectWithWhere(ctx, aWhereClause, columns);
}
```

Adding Unique Tag Identifiers and Descriptors to a Standard Table

The Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales default tag identifiers are Batch Number, Revision Number, and Lot Number. You may have a need to extend the application database to define unique tag identifiers or descriptors.

It is recommended that the data type of any unique tag identifiers or descriptors that you add be CHAR or VARCHAR.

Note: Whenever you extend the tag attributes you must also extend the console because the templates for the APIs do not contain these extended tag attributes.

For example, if you work in the metal industry, you may want to use a custom tag identifier named Steel which has both Mill and Grade attributes. Since these are not supplied by default, you must extend the set of tables listed below to include the Steel tag identifier column in each table.

Extending Tables When Adding Unique Tag Identifiers

You must extend each of the following tables whenever you add unique tag identifiers to the application database:

- YFS_COUNT_RESULT_TAG
- YFS_COUNT_TAG
- YFS_INVENTORY_AUDIT
- YFS_INVENTORY_TAG
- YFS_ITEM_TAG - The data type to be used for this table is CHAR(2).
- YFS_MOVE_REQUEST_LINE_TAG
- YFS_ORDER_KIT_LINE_SCHEDULE
- YFS_ORDER_KIT_LINE_SCHEDULE_H
- YFS_ORDER_LINE_REQ_TAG
- YFS_ORDER_LINE_REQ_TAG_H
- YFS_ORDER_LINE_SCHEDULE

- YFS_ORDER_LINE_SCHEDULE_H
- YFS_ORDER_LINE_RESERVATION
- YFS_RECEIPT_LINE
- YFS_RECEIPT_LINE_H
- YFS_SHIPMENT_LINE_REQ_TAG
- YFS_SHIPMENT_LINE_REQ_TAG_H
- YFS_SHIPMENT_TAG_SERIAL
- YFS_SHIPMENT_TAG_SERIAL_H
- YFS_WORK_ORDER_COMP_TAG
- YFS_WORK_ORDER_COMP_TAG_H
- YFS_WORK_ORDER_TAG
- YFS_WORK_ORDER_TAG_H

Extending Tables When Adding Unique Tag Descriptors

You must extend each of the following tables whenever you add unique tag descriptors to the application database:

- YFS_COUNT_RESULT_TAG
- YFS_COUNT_TAG
- YFS_INVENTORY_TAG
- YFS_ITEM_TAG The data type to be used for this table is CHAR(2).
- YFS_ORDER_LINE_REQ_TAG
- YFS_ORDER_LINE_REQ_TAG_H
- YFS_RECEIPT_LINE
- YFS_RECEIPT_LINE_H
- YFS_SHIPMENT_LINE_REQ_TAG
- YFS_SHIPMENT_LINE_REQ_TAG_H
- YFS_SHIPMENT_TAG_SERIAL
- YFS_SHIPMENT_TAG_SERIAL_H
- YFS_WORK_ORDER_COMP_TAG
- YFS_WORK_ORDER_TAG
- YFS_WORK_ORDER_TAG_H

Adding Non-Unique Indices to a Standard Table

About this task

You can add non-unique indices to entities. You add indices to a standard Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales database table, by adding an Index element in the extension XML for that table.

To add non-unique indices to a standard table:

Procedure

1. Copy the *install_dir*/installed_data/repository/entity/extensions/Extensions.xml.sample file as *install_dir*/extensions/global/entities/*your_filename*.xml file OR modify your existing extension XML file.

2. Edit the *your_filename*.xml file to add non-unique indices as shown in the following example for each table you want to extend. For a description of the XML attributes, see the table that follows the example.

```
<!-- element exposed to create index -->
<DBSchema>
<Entities>
  <Entity TableName="REQUIRED">
  .
  .
    <Indices>
      <Index Name="REQUIRED" AllowCompressedColumns="false">
         <Column Name="REQUIRED" />
         .
         .
      </Index>
        .
        .
    </Indices>
     .
     .
   </Entity>
</Entities>
</DBSchema>
```

Attribute	Description
Entity	
TableName	Required. Name of the table for which the indices are added, For example: YFS_ITEM.
Entity/Index	
Name	Required. The name of the custom index. Name should start with a prefix EXTN_
AllowCompressedColumns	Optional. If `True`, the index is allowed to contain columns marked with CompressionSupported attribute set to True.
Entity/Index/Column	
Name	Required. The name of the column for which the index is added. Create a new Column Name for each column for which the index is added.

3. Create a new Index tag for each index you want to add to the column.
4. Extend the corresponding API templates to include the non-unique indices by following the instructions in Extending API Templates.
5. Build and deploy your extensions.

Adding Foreign Key Elements to a Standard Table

About this task

A foreign key relationship is a relationship between an extended column in any application database table and the YFS_PERSON_INFO table. You can create foreign key elements to establish relationship between an extended column and the YFS_PERSON_INFO table.

Note: Currently, the YFS_PERSON_INFO is the only table which supports a relationship with foreign key extensions within the application database.

To add foreign key elements to a standard table:

Procedure

1. Copy the *install_dir*/repository/entity/extensions/Extensions.xml.sample file as *install_dir*/extensions/global/entities/*your_filename*.xml file OR modify your existing extension XML file.
2. Edit the *your_filename*.xml file to add foreign key elements as shown in the following example for each table you want to extend. For a description of the XML attributes, see the table that follows the example.

```
<!-- element exposed to create foreign key relationship -->
<DBSchema>
 <Entities>
    <Entity TableName="REQUIRED">
     .
     .
      <!-- element exposed to create relationship with PERSON_INFO table -->
        <ForeignKeys>
          <ForeignKey ParentTableName="YFS_PERSON_INFO"
            XMLName="YFSName1" >
              <Attribute ColumnName="REQUIRED"
               ParentColumnName="PERSON_INFO_KEY" />
            </ForeignKey>
          <ForeignKey ParentTableName="YFS_PERSON_INFO"
            XMLName="YFSName2" >
              <Attribute ColumnName="REQUIRED"
               ParentColumnName="PERSON_INFO_KEY" />
          </ForeignKey>
          .
          .
        </ForeignKeys>
       .
       .
    </Entity>
   </Entities>
</DBSchema>
```

Attribute	Description
Entity	
TableName	Required. Name of the table for which the foreign key elements are added. For example: YFS_ITEM.
Entity/ForeignKeys/ForeignKey	
ParentTableName	The name of the parent table for this foreign key element. **Note:** This value must be YFS_PERSON_INFO which is the only table that currently supports foreign key relationships.
XMLName	You can specify the XML representation of the element name. It must start with the prefix of the parent entity. For example, if ParentTableName is prefixed with YFS then the XMlName must start with YFS. By default the parent table name is assumed.
Entity/ForeignKeys/ForeignKey/Attribute	
ColumnName	Specifies the extended column name of the Entity.
ParentColumnName	The column name of the YFS_PERSON_INFO that has a foreign key element relationship.

3. Create a new ForeignKey tag for each foreign key relationship you want to add.
4. Multiple foreign key elements can be related to the same parent table.

5. Extend the corresponding API templates to include the foreign key elements by following the instructions in Extending API Templates.
6. Build and deploy your extensions.

Adding Text Search Indices to a Standard Table

About this task

You can add text search indices to entities. You add text search indices to a standard application database table by adding a TSIndex element in the extension XML for that table.

To add text search indices to a standard table:

Procedure

1. Copy the *install_dir*/repository/entity/extensions/Extensions.xml.sample file as *install_dir*/extensions/global/entities/*your_filename*.xml file OR modify your existing extension XML file.
2. Edit the *your_filename*.xml file to add text search indices as shown in the following example for each table you want to extend. For a description of the XML attributes, see the table that follows the example.

```
<!-- element exposed to create index -->
<DBSchema>
  <Entities>
    <Entity TableName="REQUIRED">
     .
     .
      <TSIndices>
        <TSIndex Name="REQUIRED" >
         <Column Name="USERNAME" />
        </TSIndex>
       .
       .
      </TSIndices>
     .
     .
    </Entity>
  </Entities>
</DBSchema>
```

Attribute	Description
Entity	
TableName	Required. Name of the table for which the text search indices are added. For example: `YFS_USER`.
Entity/TSIndex	
Name	Required. The name of the text search index. For example: `YFS_TS_USER_Name`. **Note:** This value cannot exceed 18 characters.
Entity/TSIndex/Column	
Name	Required. The name of the column for which the text search index is added. You cannot define multiple columns on a text index.

3. Create a new TSIndex tag for each text search index you want to add to the column.
4. Build and deploy your extensions.

Enabling Case Insensitive Searches

About this task

You can make search operations in the application case insensitive by enabling case insensitive search for the required entity/column. This is achieved by adding the attribute CaseInsensitiveSearch in the required entity XML file.

To enable case insensitivity:

Procedure

1. Edit the required Entity XML to include the `CaseInsensitiveSearch=Y` attribute.

   Include the shadow column attribute ShadowColumnName in the Entity XML and specify a name for the shadow column. If this is left blank, the system auto generates one. However, you must specify a shadow column name if you are indexing the shadow column.

   A shadow column is then generated, which is linked to the original column that is marked as case insensitive.
2. If the agent is being run in a production environment, specify the configuration mode for the entity or column in the CaseInsensitiveSearch.Mode property in the customer_overrides.properties file. Set the value of the CaseInsensitiveSearch.Mode property to `MIXED` or `DISABLED`.

   Note: By default, case insensitivity is enabled.
3. Disable the cache for the entity for which you want to run the Case Insensitive Data Loader agent.

   Note: It is recommended that along with disabling the cache, you turn off the audits for that particular entity as well.
4. Run the Case Insensitive Data Loader agent to populate data in shadow columns.

   Note: If you skipped Step 2, skip this step, as well.
5. Set the value of the CaseInsensitiveSearch.Mode property to `ENABLED` in the customer_overrides.properties file.

   Note: If you disabled audits in Step 3, enable audits now.
6. Enable the cache for the entity or column for which the Case Insensitive Data Loader agent was run in Step 3.

   The following APIs support case insensitive searches for system defined columns:

   - getOrderList
   - getItemListForOrdering
   - getExceptionListForOrder
   - getCustomerList

   For custom columns that have been extended, the case insensitive search works if the normal search works on those columns.

   Note: This feature is applicable only for searches that use the application generated XAPIs. Searches using custom queries will not be case insensitive.

Note: Case insensitive search can be enabled only for text data type (Char/Varchar columns).

Note: Use of Complex Query in conjunction with Case Insensitive search for a column is not supported. *For more information about a way to achieve this, see the Selling and Fulfillment Foundation: Customizing APIs.*

Modifying an Entity XML

Procedure

1. Edit the required Entity XML to include the attribute CaseInsensitiveSearch under the <Attributes> tag, for required columns as shown below.

```
<Entity Description="This table stores all the exceptions raised
   by the system."
   EntityType="TRANSACTION" Extensible="Y" Module="ycp"
   Name="Inbox" Prefix="YFS_" TableName="YFS_INBOX"  XMLName="Inbox">
  <Attributes>
   ........
    <Attribute ColumnName="EXCEPTION_TYPE"
     DataType="Text-40" DefaultValue="' ' "
     Description="The type of exception."
     Name="Exception_Type" Nullable="false"
     XMLName="ExceptionType" CaseInsensitiveSearch ="Y"
     ShadowColumnName="ExceptionType_LC"/>
     ......
   <Indices>
      <Index Name="EXCEPTION_TYPE_I1">
          <Column Name="ExceptionType_LC"/>
    </Index>
</Indices>
```

2. Include the ShadowColumnName attribute and specify a name for the shadow column, as shown in the above example.

Creating Custom and Hang-off Tables

The database framework allows you to extend the application database by creating custom or hang-off tables.

A custom table is an independent table and cannot be modeled as an extension to a standard application database table.

A hang-off table is a table with a many-to-one relationship with a standard application database table.

Creating a custom or hang-off entity enables you to:

- Create a relationship between a standard table and a hang-off table.
- Invoke Extensible APIs that store and retrieve data from hang-off tables.
- Invoke dbverify for generating appropriate SQL scripts to create or alter tables for custom or hang-off entities.
- Audit item and organization tables.

Keep in mind the following which apply to the creation of custom or hang-off tables:

- You can only determine if an entity is enabled for hang-off by referencing the associated Entity Relationship Diagram (ERD) located in the *install_dir*/xapidocs/ERD directory.

- Based on the Extensions.xml file, the application does not create a foreign key constraint in the EFrame_TableChanges.sql, but the foreign key relationship is enforced.
- Currently only order, order line, work order, shipment, item, and organization tables are marked as hang-off enabled.
- Custom and hang-off table names must not start with a `Y`.
- The "Extn" part is trimmed off from the XML name of the custom and hang-off tables.
- Primary key name must not start with a `Y`.
- Primary key can be of numeric data type.
- Entity names must start with the prefix provided in the entity definition.
- The YIFApi interface does not extend APIs for custom/hang-off tables. Therefore, the APIs for these tables must be configured as services.
- Javadocs are not created for the APIs created by the infrastructure to support custom and hang-off tables.
- XSD generation and validation is not done for custom or hang-off tables.
- Every custom or hang-off entity must have a primary key.

Column Name	Data Type	Default Value
Key-Column	Key OR Any numeric data type	' ' (space)

- (Optional) A custom or hang-off entity can have the following columns described in the following table:

Column Name	Data Type	Default Value
CREATETS	TimeStamp	sysdate
MODIFYTS	TimeStamp	sysdate
CREATEUSERID	UserId	' ' (space)
MODIFYUSERID	UserId	' ' (space)
CREATEPROGID	ProgramID	' ' (space)
MODIFYPROGID	ProgramID	' ' (space)
LOCKID	Lockid	0 (zero)

Note: In DB2 database, the Date data type is generated as TIMESTAMP by the Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales framework.

Steps to Create a Custom Table

About this task

To create a custom table:

Procedure

1. Copy the *install_dir*/repository/entity/extensions/Extensions.xml.sample file as *install_dir*/extensions/global/entities/*your_filename*.xml file OR modify your existing extension XML file. For example, assume that ABC_CUSTOMER_ORDER_LINE is a custom table.

2. Edit the *your_filename*.xml file to create custom tables as shown in the following example. For a description of the XML attributes, see the table that follows the example.

```
<DBSchema>
  <Entities>
     <Entity ApiNeeded="Y/N" AuditRequired="Y" Description=""
      HasHistory="Y/N" Prefix="ABC"
      TableName="ABC_CUSTOMER_ORDER_LINE" >
      <!-- table columns -->
      <Attributes>
         <Attribute ColumnName="CREATETS" DataType="TimeStamp"
           DefaultValue="sysdate" Description="Create TimeStamp" />
         <Attribute ColumnName="MODIFYTS" DataType="TimeStamp"
           DefaultValue="sysdate" Description="Modify TimeStamp" />
         <Attribute ColumnName="CREATEUSERID" DataType="UserId"
           DefaultValue="&apos; &apos;" Description="Creating User ID" />
         <Attribute ColumnName="MODIFYUSERID" DataType="UserId"
           DefaultValue="&apos; &apos;" Description="Modifying User ID" />
         <Attribute ColumnName="CREATEPROGID" DataType="ProgramID"
           DefaultValue="&apos; &apos;" Description="Creating Program ID" />
         <Attribute ColumnName="MODIFYPROGID" DataType="ProgramID"
           DefaultValue="&apos; &apos;" Description="Modifying Program ID" />
         <Attribute ColumnName="LOCKID" DataType="Lockid"
           DefaultValue="0" Description="Lock ID" />
         <Attribute ColumnName="TABLE_KEY" DataType="Key" DefaultValue=" "
           Description="" Nullable="True/False" XMLName="TableKey" />
         .
          .
      </Attributes>
      <!-- PrimaryKey is a mandatory attribute in entity definition.
        This element can have ONLY ONE attribute element -->
      <PrimaryKey Name="TABLE_NAME_PK">
           <Attribute ColumnName="TABLE_KEY" />
      </PrimaryKey>
      <!-- Indices -->
      <Indices>
          <Index Name="INDEX_I1" Unique="True/False">
              <Column Name="Attribute2" />
              .
              .
           </Index>
           .
           .
       </Indices>
       <!-- Relationship -->
       <Parent ParentTableName="YFS_ORDER_LINE" XMLName="YFSOrderLine" >
       <Attribute ColumnName="CUSTOM_ORDER_KEY"
         ParentColumnName="ORDER_LINE_KEY" />
       .
       .
       </Parent>
       <!-- ForeignKeys -->
       <ForeignKeys>
         <ForeignKey ParentTableName="PARENT_ORDER_LINE"
           XMLName="PARENTName1" >
             <Attribute ColumnName="CUSTOM_ORDER_KEY"
               ParentColumnName="PARENT_COLUMN_KEY" />
             .
             .
         </ForeignKey>
         .
         .
       </ForeignKeys>
       <!-- AuditReferences -->
       <AuditReferences>
         <Reference ColumnName="TABLE_KEY" />
```

```
            .
            .
        </AuditReferences>
      </Entity>
     </Entities>
   </DBSchema>
```

3. The following table explain the attributes in the entity XML:

Attribute	Description
Entity	
ApiNeeded	Indicate whether or not APIs should be generated. Valid values are `Y` or `N`. A default set of APIs are generated if `Y` is passed. For example in the ABC_CUSTOMER_ORDER_LINE tables, the application creates the following APIs when the database extension jar file is generated: • getABCCustomerOrderLine() • getABCCustomerOrderLineList() • createABCCustomerOrderLine() • changeABCCustomerOrderLine() • deleteABCCustomerOrderLine() These APIs can be accessed as services using the Service Definition Framework.
AuditRequired	If set to `Y`, an audit record for this entity is created.
HasHistory	This flag indicates whether the custom table can have a history table associated with it. The default value is `N`. If the flag is set to `Y`, the appropriate scripts for generating database scripts for creating and altering the history table is generated by dbverify. For a custom table, the HasHistory flag must be set to `Y` for generating history tables. However, if a Parent relationship is defined in the entity XML, this flag is copied from the parent table definition, and all child entities cannot override this flag.
Prefix	The prefix added to your custom tables. It is recommended that you do not use a prefix starting with `Y`.
TableName	The name given to your custom table.
Entity/Attributes/Attribute	
ColumnName	The names of the column that comprise the table.
DataType	The data type of the column. Valid data types are given in the *install_dir*/repository/datatypes/datatypes.xml file.

Attribute	Description
CompressionSupported	Optional. Attribute used to indicate whether or not the data compression is supported for this column. Valid values are `True` or `False`. If `True` compression support is enabled. **Note:** If the data inserted into a column has both CompressionSupported and UseCompression attributes set to `True`, then this attribute should not be set back to `False`. Doing so would result in all compressed data being retrieved without any decompression.
UseCompression	Optional. Attribute used to compress data for this column. Valid values are `True` or `False`. If `True` data is compressed. **Note:** The value of this attribute should be set to `True` only if the CompressionSupported attribute is set to `True`.
QueryAllowed	Optional. Attribute used to enable a compressible column to be used for queries in a List API. Valid values are `True` or `False`. If `True` the column can be used for queries in the List APIs. **Note:** If the value of the CompressionSupported attribute is set to true then the value of this attribute should also be set to `True`.
DefaultValue	Default value for the column.
Description	A description of the columns that could be used in Javadocs or ERD.
Nullable	Optional. Attribute used to describe the nullable value of a field. Default is false. `Nullable=true` is allowed for all columns except Primary Key Attributes and Entity Relationships.
XMLName	Optional. XML name of the attribute, if it is different from the name of the attribute. Choose a name that does not conflict with the base extension. It is recommended that you use `Extn` as a prefix. It is also strongly recommended that you use the same convention for arriving at the XMLName as the base product does: Make each letter following the underscore in the column name upper case, and the rest lower case. Then, remove the underscores. Thus, `Extn_Item_Id` should be: `ExtnItemId`.
Entity/PrimaryKey	
Name	Name of the unique index created for the primary key. This value cannot exceed 18 characters. **Note:** The name of the primary key in the extension XML should end with `_PK`.
ColumnName	The name of the table column that is identified as the primary key.
Entity/Indices/Index	

Attribute	Description
Name	The index name. This value cannot exceed 18 characters.
Unique	This key is present only for custom entities. Valid values are `True` or `False`. If `True` a unique index is created.
AllowCompressedColumns	Optional. If `True`, the index is allowed to contain columns marked with CompressionSupported attribute set to `True`.
Column/ Name	The table column name associated with the index.
Entity/Parent	
ParentTableName	Name of the other table this entity has foreign key relationship.
XMLName	The XML name of the parent attribute. It should start with the prefix mentioned in the parent table. By default the parent table name is assumed.
Parent/Attribute Level	
ParentColumnName	Column name in the parent table. Note: To create relationships among entities, the data type of parent column must be of type `CHAR` or `VARCHAR`.
ColumnName	Column name in this custom entity.
Entity/ForeignKeys/ForeignKey	
ParentTableName	The name of the table with which the entity has a foreign key relationship.
XMLName	XML representation of the element name. By default the parent table name is assumed.
Entity/ForeignKeys/ForeignKey/Attribute	
ParentColumnName	Column name of the parent table. Note: To create foreign keys among entities, the data type of parent column must be of type `CHAR` or `VARCHAR`.
ColumnName	Column name in this custom entity.
Entity/AuditReferences/Reference	
ColumnName	Reference Column name in the audit table.

Note: In entity definition, relationship can be defined under Parent and ForeignKey elements.

4. The relationship defined under the ForeignKey element indicates:
   a. If the foreign table is an application database table, for a single record in the foreign table, zero or many records in this custom table may exist.
   b. This is a read-only relationship, hence deletion of a record from the foreign table does not result in the deletion of a matching record from this custom table.

5. The relationship defined under the `Parent` element indicates:
   a. For a single record in the parent table, multiple child records may exist.
   b. Deletion of a record from the parent table results in the deletion of matching records from the child table, if any.
6. Extend the corresponding API templates (for example, `getOrderDetails()`API) by following the instructions in Extending API Templates.

   Note: The APIs generated by the application for the custom tables can be invoked as a service and through a multiAPI wrapper component.
7. Build and deploy your extensions.

Steps to Create a Hang-off Table

About this task

To create a hang-off table:

Procedure

1. Copy the *install_dir*/repository/entity/extensions/Extensions.xml.sample file as *install_dir*/extensions/global/entities/*your_filename*.xml file OR modify your existing extension XML file. For example, assume that ABC_CUSTOMER_ORDER_LINE is a hang-off table.
2. Edit the *your_filename*.xml file to create hang-off tables as shown in the following example. For a description of the XML attributes, see the table that follows the example.

```
<DBSchema>
   <Entities>
     <Entity ApiNeeded="Y/N" AuditRequired="Y" Description=""
        HasHistory="Y/N" Prefix="ABC"
        TableName="ABC_CUSTOMER_ORDER_LINE" >
        <!-- table columns -->
        <Attributes>
            <Attribute ColumnName="CREATETS" DataType="TimeStamp"
             DefaultValue="sysdate" Description="Create TimeStamp" />
            <Attribute ColumnName="MODIFYTS" DataType="TimeStamp"
             DefaultValue="sysdate" Description="Modify TimeStamp" />
            <Attribute ColumnName="CREATEUSERID" DataType="UserId"
             DefaultValue="&apos; &apos;" Description="Creating User ID" />
            <Attribute ColumnName="MODIFYUSERID" DataType="UserId"
             DefaultValue="&apos; &apos;" Description="Modifying User ID" />
            <Attribute ColumnName="CREATEPROGID" DataType="ProgramID"
             DefaultValue="&apos; &apos;" Description="Creating Program ID" />
            <Attribute ColumnName="MODIFYPROGID" DataType="ProgramID"
             DefaultValue="&apos; &apos;" Description="Modifying Program ID" />
            <Attribute ColumnName="LOCKID" DataType="Lockid" DefaultValue="0"
             Description="Lock ID" />
            <Attribute ColumnName="TABLE_KEY" DataType="Key" DefaultValue=" "
             Description="" Nullable="True/False" XMLName="TableKey" />
             .
             .
        </Attributes>
        <!-- PrimaryKey is a mandatory attribute in entity definition. This
          element can have ONLY ONE attribute element -->
        <PrimaryKey Name="TABLE_NAME_PK">
           <Attribute ColumnName="TABLE_KEY" />
        </PrimaryKey>
        <!-- Indices -->
        <Indices>
            <Index Name="INDEX_I1" Unique="True/False">
                <Column Name="Attribute2" />
                .
```

```
            .
         </Index>
         .
         .
      </Indices>
      <!-- Relationship -->
      <Parent ParentTableName="YFS_ORDER_LINE" XMLName="YFSOrderLine" >
         <Attribute ColumnName="CUSTOM_ORDER_KEY"
        ParentColumnName="ORDER_LINE_KEY" />
        .
        .
      </Parent>
      <ForeignKeys>
         <ForeignKey ParentTableName="PARENT_ORDER_LINE"
           XMLName="PARENTName1" >
            <Attribute ColumnName="CUSTOM_ORDER_KEY"
              ParentColumnName="PARENT_COLUMN_KEY" />
             .
             .
         </ForeignKey>
         .
         .
         </ForeignKeys>
      <!-- AuditReferences -->
      <AuditReferences>
        <Reference ColumnName="TABLE_KEY" />
        .
        .
      </AuditReferences>
   </Entity>
  </Entities>
</DBSchema>
```

3. The following table explains the attributes in the entity XML:

Attribute	Description
Entity	
ApiNeeded	Indicates whether or not APIs should be generated. Valid values are `Y` or `N`. A default set of APIs are generated if `Y` is passed. For example in the ABC_CUSTOMER_ORDER_LINE table, the application creates the following APIs when the database extension jar file is generated: • listABCCustomerOrderLine() • getABCCustomerOrderLine() • createABCCustomerOrderLine() • modifyABCCustomerOrderLine() • deleteABCCustomerOrderLine() These APIs can be accessed as services using the Service Definition Framework.
AuditRequired	If set to `Y`, audit record for this entity is created. Note: This attribute must not be passed when you are creating a hang-off for order related tables. In this case, the audits are automatically inserted into the YFS_ORDER_AUDIT table.
Description	A description of the entity that could be used in Javadocs or ERD.

Attribute	Description
HasHistory	This flag is automatically inherited from the parent table. For example, let us assume that ABC_ORDER_HEADER table is created as an hang-off table for YFS_ORDER_HEADER, which has an associated history table. Then ABC_ORDER_HEADER_H is automatically generated by the database framework.
Prefix	The prefix added to your custom tables. It is recommended that you do not use a prefix starting with `Y`.
TableName	The name given to your hang-off table.
Entity/Attributes/Attribute	
ColumnName	The names of the column that comprise the table.
DataType	The data type of the column. Valid data types are given in *install_dir*/repository/datatypes/datatypes.xml file.
CompressionSupported	Optional. Attribute used to indicate whether or not the data compression is supported for this column. Valid values are `True` or `False`. If `True` compression support is enabled. **Note:** If the data inserted into a column has both CompressionSupported and UseCompression attributes set to `True`, then this attribute should not be set back to `False`. Doing so would result in all compressed data being retrieved without any decompression.
UseCompression	Optional. Attribute used to compress data for this column. Valid values are `True` or `False`. If `True` data is compressed. **Note:** The value of this attribute should be set to `True` only if the CompressionSupported attribute is set to `True`.
QueryAllowed	Optional. Attribute used to enable a compressible column to be used for queries in a List API. Valid values are `True` or `False`. If `True` the column can be used for queries in the List APIs. **Note:** If the value of the CompressionSupported attribute is set to true then the value of this attribute should also be set to `True`.
DefaultValue	Default value for the column
Description	A description of the columns that could be used in Javadocs or ERD.
Nullable	Optional. Attribute used to describe the nullable value of a field. Default is false. `Nullable=true` is allowed for all columns except Primary Key Attributes and Entity Relationships.
XMLName	Optional. XML name of the attribute, if it is different from the name of the attribute. Choose a name that does not conflict with the base extension. It is recommended that you use `Extn` as a prefix. It is also strongly recommended that you use the same convention for arriving at the XMLName as theSterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales base product does: Make each letter following the underscore in the column name upper case, and the rest lower case. Then, remove the underscores. Thus, `Extn_Item_Id` should be: `ExtnItemId`.
Entity/PrimaryKey	

Attribute	Description
Name	Name of the unique index created for the primary key. This value cannot exceed 18 characters. **Note:** The name of the primary key in the extension XML should end with `_PK`.
ColumnName	The name of the table column that is identified as the primary key.
Entity/Indices/Index	
Name	The index name. This value cannot exceed 18 characters.
Unique	This key is present only for custom entities. Valid values are `True` or `False`. If `True` a unique index is created.
AllowCompressedColumns	Optional. If `True`, the index is allowed to contain columns marked with CompressionSupported attribute set to `True`.
Column/ Name	The table column name associated with the index.
Entity/Parent	
ParentTableName	Name of the other table this entity has foreign key relationship.
XMLName	The XML name of the parent attribute. It should start with the prefix mentioned in the parent table. By default the parent table name is assumed.
Parent/Attribute Level	
ParentColumnName	Column name in the parent table. Note: To create relationships among entities, the data type of parent column must be of type `CHAR` or `VARCHAR`.
ColumnName	Column name in this custom entity.
Entity/ForeignKeys/ForeignKey	
ParentTableName	The name of the table with which the entity has a foreign key relationship.
XMLName	XML representation of the element name. By default the parent table name is assumed.
Entity/ForeignKeys/ForeignKey/Attribute	
ParentColumnName	Column name of the parent table. Note: To create foreign keys among entities, the data type of parent column must be of type `CHAR` or `VARCHAR`.
ColumnName	Column name in this hang-off entity.
Entity/AuditReferences/Reference	
ColumnName	Reference Column name in the audit table.

Note: In entity definition, relationship can be defined under ForeignKey elements.

4. The relationship defined under the ForeignKey element indicates:
   a. If the foreign table is aSterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales table, for a single record in the foreign table, zero or many records in this hang-off table may exist.

b. This is a read-only relationship, hence deletion of a record from the foreign table does not result in the deletion of a matching record from this hang-off table.

5. Extend the corresponding API templates (for example, getOrderDetails API) by following the instructions in "Extending API Templates."

   Note: The APIs generated for the hang-off tables can be invoked as a service and through a multiAPI wrapper component.

6. Build and deploy your extensions.

Purging Data from Hang-off Tables

Currently, the Purge agent moves records to history tables. With the hang-off entities enabled, the Purge agent also deletes records from hang-off tables. However, the data from a hang-off table can be purged only if its parent elements are also purged. If a history table exists, records are added to the history table. The records are deleted from the history table using the History Purge agent.

In order to purge the hang-off entities you need to include the entities.jar file in the classpath of the agent server.

Chapter 4. Extending the Data Types Files

Extending the Data Types Files

You can extend the attributes available to you by adding your own XML attributes and abstract data types to the datatypes.xml file.

You will need to modify the datatypes.xml file in the following cases:

- When you want to change the actual size of an existing data type.
- When you want to restrict the length of the input for a particular field on the UI. For example, say the data type allows the input to be of length 20 but in the UI you want to restrict the length of the input to just 10.
- When the existing data types does not meet your requirements and you want to create a completely new data type.

You will need to modify the yfsdatatypemap.xml file in the following case:

- When you have defined a completely new data type in the datatypes.xml file and you want to define the mapping for this new data type in the yfsdatatypemap.xml file.

Extending the Data Type Map File

About this task

To extend the data type map XML file:

Procedure

1. Create a new *install_dir*/extensions/global/template/resource/yfsdatatypemap.xml file.

   If the /global/template/resource/ directory structure does not exist, create the required directory structure.
2. Add an XML root node in the same way it appears in the *install_dir*/repository/xapi/template/merged/resource/yfsdatatypemap.xml file.
3. Add any attributes that need to be mapped in the yfsdatatypemap.xml file.
4. Build and deploy your extensions.

Extending the Data Type File

About this task

To extend the data type XML file:

Procedure

1. Create a new *install_dir*/extensions/global/etc/datatypes.xml file, if it does not already exist.

   If the /global/etc/ directory structure does not exist, create the required directory structure.

   Note: When datatypes are extended through *install_dir*/extensions/global/etc/datatypes.xml file, the resources.jar file needs to be rebuilt before running the dbverify tool.

2. Add an XML root node in the same way it appears in the *install_dir*/repository/datatypes/datatypes.xml file.
3. Add any differential values for the datatypes, including the following:
   - Add and define parameters for new datatypes
   - Modify parameters of existing datatypes

   Note: For existing datatypes, you can modify only the UI related attributes in the datatypes.xml file such as UI Size and UITableSize.

   Note: You cannot resize the Date input fields within the console across the board even if your date format is larger than the default date format used by the application.

   Note: The application reserves the Type attribute for internal use, and so you cannot override it. All other attributes can be overridden.

   Note: For existing datatypes, you cannot change a character type to a numeric type. For example, you cannot change a CHAR type to NUMBER type. An exception will be thrown for such invalid conversions. Following conversions are treated as invalid conversions:
   - Converting NVARCHAR, NCHAR, VARCHAR2, BLOB, CLOB, TIME, DATE, DATETIME type to NUMBER type.
   - Converting NVARCHAR, NCHAR, VARCHAR2, BLOB, CLOB, TIME, DATE, DATETIME type to SCI_LONG type.
   - Converting NVARCHAR, NCHAR, VARCHAR2, BLOB, CLOB, TIME, DATE, DATETIME type to SCI_INT type.
   - Converting any type to BLOB, CLOB type.
   - Converting BLOB, CLOB type to any type.
   - Converting from any type to DATE, DATETIME, TIME type.

   Note: The invalid conversions are not specific to any database. The invalid conversions are based on the criteria whether the new datatype can hold the data that the old datatype column can take.

   Note: If you add any new table column extensions and you are using SQLServer, ensure that they have NCHAR and NVARCHAR character-based datatypes.Do not use CHAR and VARCHAR datatypes for SQLServer.
4. Build and deploy your extensions.

Chapter 5. Create a Custom View

Create a Custom View to Join Multiple Tables

About this task

A custom view can be created to join multiple DB2 tables. This approach can be used for either out-of-the-box or custom tables.

Product APIs are then used to query from the custom views. Custom queries are not required.

Procedure

1. Create custom table(s), if required.
2. Write a View entity, using the example below as a guideline.

   The db framework generates db classes and APIs on the view to retrieve records.

   Restrictions: Classes or APIs generated in this way support the get and list API functions only, and cannot be used to modify data.

   dbverify does not process or handle views. Any change made in the xml file for a view is not processed by dbverify.
3. Customize the appropriate console to invoke the database APIs corresponding to the view.
4. Build and deploy your customization.

Example: Inventory Item Search

A business case necessitates an inventory item search (inventory console) based on extended attributes of yfs_item. Such a search requires the yfs_item and yfs_inventory_item tables to be joined in a view, but yfs_inventory_item is not extensible.

The view entity is defined as shown in the ExtnCatalogSearchConfigProperties.xml file below, and the inventory console is customized to invoke the database APIs corresponding to the view.

The following is the sample ExtnCatalogSearchConfigProperties.xml file:

```
<DBSchema>
  <Entities>
    <Entity TableName="EXTN_INV_ITEM_VW"
            Description="This view joins YFS_ITEM and YFS_INVENTORY_ITEM
        tables to enable querying based on webclass and subclass
        attributes"
            View="true"
            EntityType="VIEW"
            HasHistory="False"
            AuditRequired="N"
            ApiNeeded="Y"
            Prefix="EXTN" >
      <Attributes>
        <Attribute ColumnName="ITEM_INV_VW_KEY"
                   DataType="Key"
```

```
                DefaultValue="' '"
                Description="Primary key for this view."
                Nullable="false"
                XMLName="ItemInVwKey"/>
      <Attribute ColumnName="ITEM_ID"
                DataType="ItemID"
                Description="Identifer for this inventory item."
                Name="Item_Id"
                Nullable="false"
                XMLName="ItemID"
                DefaultValue="' '" />
      <Attribute ColumnName="UOM"
                DataType="UOM"
                Description="Unit of measure for this inventory item."
                Name="Uom"
                Nullable="false"
                XMLName="UnitOfMeasure"
                DefaultValue="' '" />
      <Attribute ColumnName="EXTN_SUBCLASS"
                DataType="VARCHAR2-24"
                Type="VARCHAR2"
                Size="24"
                DefaultValue="' '"
                Nullable="false"
                XMLName="ExtnSubclass"/>
      <Attribute ColumnName="DEFAULT_PRODUCT_CLASS"
                DataType="ProductClass"
                DefaultValue="' '"
                Description="Default product class of an item."
                Name="Default_Product_Class"
                Nullable="false"
                XMLName="DefaultProductClass"/>
      <Attribute ColumnName="PRODUCT_CLASS"
                DataType="ProductClass"
                DefaultValue="' '"
                Description="Product class for the item of this
                  inventory audit."
                Name="Product_Class"
                Nullable="false"/>
      <Attribute ColumnName="DESCRIPTION"
                DataType="ItemDesc"
                DefaultValue="' '"
                Description="Description of the item."
                Name="Description"
                Nullable="false"
                XMLName="Description"/>
    </Attributes>
    <PrimaryKey>
      <Attribute ColumnName="ITEM_INV_VW_KEY"
                Name="Item_Inv_Vw_Key"/>
    </PrimaryKey>
  </Entity>
 </Entities>
</DBSchema>
```

Chapter 6. Generating Audit References for Entities

About Extending Audit References

About this task

If the AuditRequired flag is enabled in the entity XML, audit records are added to the YFS_AUDIT table. The default for this flag is Y, for item and organization tables. However, the audit flag and audit references can be overridden by the extension XML file.

Note: All the records pertaining to the cached tables as well as the tables for which the value of AuditRequired flag is set to Y are logged into YFS_AUDIT table.

If you want to switch off the generation of audit references for some specific entities, change the value of the AuditRequired flag to N for such entities.

Note: You can add new audit references in the extension XML file. When new references are added, they take precedence over the existing audit references, which are entirely overridden.

You can add up to six audit references only.

Only item and organization header-level audit records are inserted in the YFS_AUDIT_HEADER table. The audit references refer to the columns of the entity being audited.

The audits can be generated for the hang-off and custom tables, by modifying the entity table name and audit reference column names.

Note: Auditing is not supported for hang-off tables with more than one parent.

To generate audit references for entities:

Procedure

1. Edit the *your_filename*.xml file in the *install_dir*/repository/entity/extensions directory to enable audit record generation for desired entities. The following example explains the elements to be added to the database schema:

```
<DBSchema>
   <Entities>
     <Entity TableName="YFS_ITEM" AuditRequired="Y" >
        .
        .
        <AuditReferences>
            <Reference ColumnName="ItemId" />
            .
            .
        </AuditReferences>
        .
        .
     </Entity>
   </Entities>
</DBSchema>
```

Attribute	Description
Entity	
TableName	The table name to be audited.
AuditRequired	If this flag is set to Y the audit references are entered in the YFS_AUDIT table. **Note:** This attribute must not be passed when you are creating a hang-off for order related tables. In this case, the audits are automatically inserted into the related order audit tables.
Entity/AuditReferences/Reference	
ColumnName	The column name in this entity which has audit references. This name must be valid for the entity.

2. Create a new Reference tag for each audit reference you want to add.
3. The hang-off of an order table audits can be viewed with the associated order audits.

Chapter 7. Extending API Templates

About Extending API Templates

About this task

Each template-based API delivers different output, depending on the template passed to it. To verify whether an API is template-based or not, see the Javadocs.

If your table modifications impact any APIs, you must extend the templates of those APIs. Place the extended API templates in the *install_dir*/extensions/global/template/api directory.

To find out which APIs are impacted by table modifications:

Procedure

1. Note the XMLName attribute of the table being modified in the entity tag inside the database entity XML files (which contains the definition of all the tables). These database entity XML files are located in *install_dir*/repository/entity directory.
2. Search for the pattern of that XMLName attribute in the *install_dir*/extensions/global/template/api directory. The search results in finding exposed and internal APIs impacted by the table modifications or extensions.

   For example, consider that you want to extend an attribute in the YFS_CHARGE_CATEGORY table. The XMLName for this table as specified in *install_dir*/repository/entity/omp_tables.xml is ChargeCategory. Now search for the attribute ChargeCategory in *install_dir*/extensions/global/template/api directory to find the APIs impacted by this extension.

Including Extended Attributes in the API Template

The extended attributes appear as a separate `<Extn>` element under the primary element.

For example, in the default output XML template of the getItemDetails() API, the Item attributes have the following structure:

```
<?xml version="1.0" encoding="UTF-8"?>
 <Item .. Item attributes >
    <PrimaryInformation .... PrimaryInformation attributes />
    <ItemServiceSkillList .. ItemServiceSkillList attributes/>
    <ItemAliasList ... ItemAliasList attributes />
   .
   .
 </Item>
```

After extending the Item header, the getItemDetails() API can output the following XML:

```
<?xml version="1.0" encoding="UTF-8"?>
 <Item .. Item attributes >
   <PrimaryInformation .... PrimaryInformation attributes />
   <Extn ExtnAltQty="200408201034469490" ..... extnded attributes />
   <YFSPersonInfo .... PersonInfoKey="200408201034469490" ...../>
   <ItemServiceSkillList .. ItemServiceSkillList attributes/>
```

```
  <ItemAliasList ... ItemAliasList attributes />
  .
  .
</Item>
```

Note: Foreign Key variables for the extended column appear as a PersonInfoKey attribute of the YFSPersonInfo element. The relationship can be validated if the extended column and the PersonInfoKey have the same value.

The extended attribute is retrieved from the XMLName attribute of the *your_filename*.xml file that you edited in the previous sections, when extending a standard table. Place your extended templates in the *install_dir*/extensions/global/template/api directory.

Including Custom and Hang-Off Entities in the API Template

About this task

The standard APIs can be extended to provide information from the custom or hang-off tables. A tool specifically provided for generating the template XML's, templateXmlGen.xml is located in the *install_dir*/bin directory.

Procedure

1. Run the template XML generation tool from your *install_dir* directory by using the following command:

   ```
   sci_ant.sh -Dtable=TABLE_NAME -f bin/templateXmlGen.xml
   ```

2. Once the command is run, the sample XML files are placed in the *install_dir*/extn/sampleXML directory as *TABLE_NAME*_sample.xml.

   For example, consider HF_Order_Header is a hang-off of YFS_Order_Header table. The generated HF_Order_Header_sample.xml is as follows:

   ```
   <HFOrderHeader Createprogid=" " Description=" " DocumentType=" "
   EnterpriseKey=" " OrderHeaderKey=" " OrderName=" " OrderNo=" " .... >
   </HFOrderHeader>
   ```

3. A sample XML for including the above attributes in a standard API can be generated by passing the YFS table that has a relationship with the hang-off table you are interested in.

   For example, assume HF_Order_Header is a hang-off table with a relationship to the YFS_Order_Header table. The XML template generated by the tool when TABLE_NAME=YFS_Order_Header is passed:

   ```
   <Order>
     <OrderLines>
        <OrderLine ........>
          <Extn extended attributes >
            <HFOrderHeaderList>
              <HFOrderHeader Createprogid=" " Description=" " .........>
              </HFOrderHeader>
            </HFOrderHeaderList>
          </Extn>
        </OrderLine>
     </OrderLines>
   </Order>
   ```

   Note: You can modify the attributes only within your custom or hang-off element.

You can prune this sample XML to include your custom attributes in an API template, such as getOrderDetails output template. However, you cannot modify any of the YFS elements or attributes.

Note: The sample XMLs are also automatically generated when you create the database extension jar file, and are posted in the *install_dir*/xapidocs/sampleXML directory. However, if you need to create a sample template, you must run the template XML generation tool separately by specifying the corresponding YFS table name.

4. A hang-off table can be deleted by passing an Operation attribute in the change or modify APIs. For example, HF_Order_Header element can be deleted in a changeOrder API as:

```
<Order>
  <OrderLines>
    <OrderLine ........>
        <Extn extended attributes >
           <HFOrderHeaderList>
             <HFOrderHeader Operation="Delete" Createprogid=" " ....... >
             </HFOrderHeader>
           </HFOrderHeaderList>
        </Extn>
     </OrderLine>
  </OrderLines>
</Order>
```

   The operations such as Create and Modify are run by default. If an entry for that element exists, the API modifies the entries with the recent value. In the case where that element does not exist it creates a new entry.

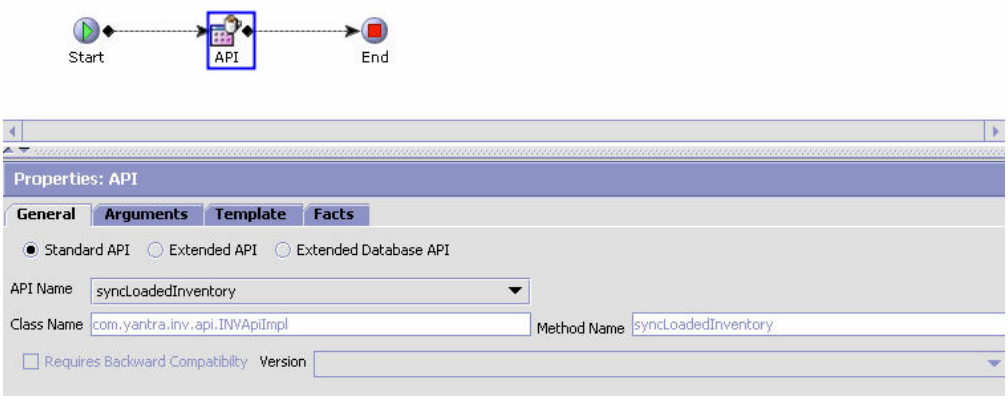
5. The records in a hang-off table can be reset by assigning the value of “true” to the Reset attribute at the list level element of the XML. When the records are reset, all existing records for that hang off table that correspond to the parent table are deleted and all elements included under the list element are inserted. For example, hang off records in the HF_Order_Header_list element can be reset by using the following:

```
<Order>
   <OrderLines>
     <OrderLine ........>
         <Extn extended attributes >
            <HFOrderHeaderList Reset="true">
               <HFOrderHeader>
               </HFOrderHeader>
            </HFOrderHeaderList>
         </Extn>
     </OrderLine>
   </OrderLines>
 </Order>
```

6. Build and deploy your extensions.

Configuring Services for Custom and Hang-off APIs

The APIs generated for custom or hang-off entities by the application can be invoked as a service and through a multiAPI wrapper component. For more information on invoking the APIs through a multiAPI component, refer to the Javadocs. The service configuration user interface has to be enabled to configure these APIs.



To include custom APIs, you can create a service definition as shown in the figure. The configuration fields are explained in the following table.

Field Name	Description
General Tab	
Standard API	Select this option if a standard application API is to be invoked. If selected, the standard API Name drop down list displays. For each API, the Class Name and Method Name are provided and cannot be edited.
Extended API	Select this option if a custom Java code is to be invoked.
Extended Database API	Select this option if the service invokes a custom or hang-off API. If selected, a custom API Name drop-down list displays. For each API, the Class Name and Method Name are provided and cannot be edited. **Note:** If you want to lock a record in a custom table, pass the SelectMethod attribute as part of the input XML to the custom entity API. The locking happens within the transaction boundary of the custom API call. The SelectMethod attribute can take the following values: • WAIT • NO_WAIT • NONE
API Name	Select or enter the API to be called. **Note:** This field is for integration purposes only.
Class Name	Specifies the class to be called.
Method Name	Specifies the method to be called.
Requires Backward Compatibility	Select this field to indicate that input data coming through the API is from a previous version (only applicable to system APIs).
Version	If you chose Requires Backward Compatibility, select the application version the API is to behave as. Only the applicable versions for the individual API display.
Arguments Tab	

Field Name	Description
Argument Name	You can pass name/value pairs to the API by entering the values in the Arguments Tab. In order for custom APIs to access custom values, the API should implement the interface com.yantra.interop.japi.YIFCustomApi. If entered, these name/value pairs are passed to the CustomApi as a Properties object.
Argument Value	Enter the argument value.
Template Tab	
	When system APIs are invoked, you can specify an output template to be used by the API. You can specify the template in the configuration properties of the Service Definition, the Resource Definition in the Resource Hierarchy tree, or both. However, if the template has been specified at both definition levels, the template specified in the Service Definition is used.
XML Template	Select this radio button to construct the XML to be used for the API output. Enter the template root element name and click OK. You can then construct the XML.
File Name	Select this radio button to enter the filename of the XML file to be used as the API output template. This file should also exist in your CLASSPATH.
Facts Tab	
	A Fact is an attribute that is used by an API or an agent to identify which colony to connect to and retrieve data from. Based on the fact name and fact value entered, the corresponding colony is determined.
Fact Name	Enter the fact name of the XML attribute.
Fact Value	Enter the fact value of the XML attribute.

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing

IBM Corporation

North Castle Drive

Armonk, NY 10504-1785

U.S.A.

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing

Legal and Intellectual Property Law

IBM Japan Ltd.

19-21, Nihonbashi-Hakozakicho, Chuo-ku

Tokyo 103-8510, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:
INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be

incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation

J46A/G4

555 Bailey Avenue

San Jose, CA 95141-1003

U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

All IBM prices shown are IBM's suggested retail prices, are current and are subject to change without notice. Dealer prices may vary.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© IBM 2013. Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. 2013.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Trademarks

IBM, the IBM logo, and ibm.com® are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at “Copyright and trademark information” at http://www.ibm.com/legal/copytrade.shtml.

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

IT Infrastructure Library is a registered trademark of the Central Computer and Telecommunications Agency which is now part of the Office of Government Commerce.

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

ITIL is a registered trademark, and a registered community trademark of the Office of Government Commerce, and is registered in the U.S. Patent and Trademark Office.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Cell Broadband Engine is a trademark of Sony Computer Entertainment, Inc. in the United States, other countries, or both and is used under license therefrom.

Linear Tape-Open, LTO, the LTO Logo, Ultrium and the Ultrium Logo are trademarks of HP, IBM Corp. and Quantum in the U.S. and other countries.

Connect Control Center®, Connect:Direct®, Connect:Enterprise®, Gentran®, Gentran®:Basic®, Gentran:Control®, Gentran:Director®, Gentran:Plus®, Gentran:Realtime®, Gentran:Server®, Gentran:Viewpoint®, Sterling Commerce™, Sterling Information Broker®, and Sterling Integrator® are trademarks or registered trademarks of Sterling Commerce®, Inc., an IBM Company.

Other company, product, and service names may be trademarks or service marks of others.

Product Number: xxxx-xxx

Printed in USA