Sterling Selling and Fulfillment Foundation

Extending Transactions

Version 9.1

Sterling Selling and Fulfillment Foundation

Extending Transactions

Version 9.1

Note
Before using this information and the product it supports, read the information in "Notices" on page 29.

Contents

Chapter 1. Checklist for Customization Projects

Customization Projects

Projects to customize or extend Sterling Selling and Fulfillment Foundation 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.

The items identified for extension and/or modification in the documentation are Source Components (to the extent such item involves source code) and Sample Materials for purposes of the License Information file associated with this product.

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. Extending Transactions

About Extending Transactions

You can extendSterling Selling and Fulfillment Foundation programmatically both to enhance the functionality of your implementation and to integrate with external systems.

This section describes how to achieve extensibility programmatically, using these mechanisms. Transaction processing mechanisms in the application can be classified into two basic categories:

- Synchronous (on demand) or asynchronous (message driven) services
- Time-triggered transactions

About Time Triggered Transactions

A time-triggered transaction, or agent, is a program that performs a variety of individual functions, automatically and at specific time intervals. It is not triggered by conditions, events, or user input.

There are three types of time-triggered transactions:

- Business process transactions - responsible for processing day-to-day transactions
- Monitors - watch and send alerts for processing delays and exceptions
- Purges - clear out data that may be discarded after having been processed

You can extend the transactions provided by the application by using one of the following mechanisms:

- User exits
- Event handlers

For information on using the time-triggered transactions provided by your application, see the Sterling Selling and Fulfillment Foundation: Configuration Guide.

Using User Exits to Extend Standard Transactions

Sterling Selling and Fulfillment Foundation provides the ability to extend or override key business algorithms. This is accomplished through user exits that are invoked when the algorithms are run.

A typical user exit can:

- Override application logic by providing its own logic.
- Extend application logic by providing more inputs to the application algorithm.

For example, if an order is split into multiple shipments, you may need to compute the order price differently for each shipment. In order to change the way Sterling Selling and Fulfillment Foundation computes order prices, you can override the specific computation or algorithm in the repriceOrder user exit.

Each user exit is a separate Java interface. You may choose to implement only those user exits where you want to override or augment the business logic.

Note: When the API or time-triggered transaction is executing the default algorithm, the user exit is called. If your implementation of the function throws an exception, the current transaction is rolled back.

For detailed descriptions of each user exit, see the following Javadocs packages*:*

- com.yantra.ycm.japi.ue
- com.yantra.ycp.japi.ue
- com.yantra.ydm.japi.ue
- com.yantra.yfs.japi.ue

Implementing and Deploying User Exits

All user exit classes should be deployed as a JAR file that is available in the CLASSPATH of the agent adapter script and in the smcfs.ear file deployed in your application server. For more information about implementing user exits, see the Sterling Selling and Fulfillment Foundation: *Configuration Guide*.

For information on creating and deploying custom classes, see the Sterling Selling and Fulfillment Foundation :*Installation Guide*.

Guidelines for User Exits

The following guidelines have to be kept in mind when you are using User Exits within the application API:

- User Exits are structured to return specific information and hence their usage must be restricted for such purpose alone.
- From the user exits, you cannot invoke APIs that modify the data. This is to ensure that errors do not occur because of the data that is getting modified in the parent transaction (the transaction that calls the user exit), and the same data getting modified in the user exit custom code. For example, you cannot invoke a changeOrder() API from a user exit that obtains information related to the same order.

However, APIs that do not modify the data (like select APIs) can be invoked in the user exits. For example, you can call getOrderDetails() API from a user exit.

Using Event Handlers to Extend Standard Transactions

The application raises events at specific moments in processing. It enables you to define an action to be performed when a specific event occurs.

In the application configuration, an event is defined to invoke the application's event handler. The event handler performs special processing on data published by the event before being transported to the transport services layer.

As part of the event configuration, services can be invoked. When such services are invoked, some of the events, like INVENTORY_CHANGE pass data as a map whereas the other events pass data as an XML. In the event of data being passed as a map, when the service is configured for such an event the data map is converted to an XML as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<SterlingXML>
   <XML AccountNo="" AdjustmentType="ADJUSTMENT"
       EnterpriseCode="DEFAULT" ItemID="ITEM1"
       ItemKey="2005030116023851364" ..... />
</SterlingXML>
```

Note: By default, when you raise an event that has Map data, the data map gets converted to an XML with 'SterlingXML' as the root tag. If you want to override this root tag value, you must set the yfs.sci.event.flow.roottag property value as the new root tag in the *install_dir*/properties/customer_overrides.properties file.

About Configuring Events

In Release 5.0 and later, you can associate an action to a service to perform special processing required. For example, the event PUBLISH_SHIP_ADVICE could invoke the application event handler to call a custom Java class. The Java class can augment the publish ship advice XML. The Service Definition Framework can transport the enriched XML data to the transport services layer.

For more information on configuring actions and associating the actions to events, or on the various components you use to build a service, see the Sterling Selling and Fulfillment Foundation: *Configuration Guide*.

The event handlers that can be invoked from an action (configured in the Other tab) are described below.

Note: The following event handlers are provided exclusively for backward compatibility purposes.

E-Mail Message Event Handler

The application provides a standard Send E-mail event handler for sending e-mail messages on various events. For example, when a line is shipped, you can send shipment information. SMTP-based e-mail is supported for these messages. When configured in the Other tab when configuring an Action, the e-mail is sent synchronously. If the mail server is not available, an exception is thrown. For sending an event notification in the form of an e-mail message using a service, see the Sterling Selling and Fulfillment Foundation: *Configuration Guide*.

E-Mail Templates

E-mail templates enable you format e-mail messages. The E-mail node receives XML data and merges it with the XSL template you specify. You can configure an E-mail node in the Service Builder to use an e-mail template.

Create Custom E-Mail Template

About this task

To create and use a custom e-mail template:

Procedure

1. Copy the *install_dir*/repository/xapi/template/merged/email/orders_mail.xsl template file.
2. Modify the file as needed, rename it, and save it within the *install_dir*/extensions/global/template/xsl/*CUSTOM-TEMPLATE-XSL*

directory. You can save it to another directory, but using the standard directory structure supplied bySterling Selling and Fulfillment Foundation helps ensure consistency.

3. From the Service Builder, configure an E-mail node to use your custom e-mail template. In the Body Template field specify /template/xsl/*CUSTOM-TEMPLATE-XSL*.

   Note that using the *install_dir*/repository/xapi/template/merged/email/*file_name*.mlt file with Actions to accomplish the same purpose has been deprecated as of Release 5.0. Use the E-mail node and an XSL file instead.

Exception Alert Event Handler

The application supports standard event handlers sending exception alerts to the Alert Console. These exceptions are redirected to a specific user or to an exception queue for a group of users. For example, you can send an exception alert to the customer service representative queue when authorization fails so the customer service representative can contact the buyer in person. For more information about the Alert Console, see the Sterling Selling and Fulfillment Foundation : *User Guide*.

Exception Alert Templates

Exception alert templates enable you to supply additional text to alerts raised. This enables you to make error message more descriptive and easy to understand. They also provide a means of supplying a hyperlink to the resolution screens from the Alert console. For example, for any alert created for an order, shipment, or load document type, a hyperlink is created and displays in the "Created For" column on the Alert List screens. In the Exception Alert Template you can customize this hyperlink or create any other hyperlinks. The input data to the alert node is merged with the template you specify and then posted as the description of the alert raised.

Events that publish data in data buffer format use an ECT template, which are text files that contain tags. These tags are replaced by actual data at run time. All tags use the |#YFS_*tagname*| syntax.

For example, use the tag |#YFS_OrderNo| to have the actual order number appear in its place (if OrderNo is published as a part of the data buffer). Any of the data elements published in the data buffer can be used in the template.

For the exception console, there are two templates. One template is used for determining the DETAIL_DESCRIPTION field of the YFS_INBOX table. The other template is used for determining the LIST_DESCRIPTION field of the YFS_INBOX table. The templates are merged with the data published (by an event) and the resulting string is populated to the corresponding field.

Create Custom Exception Alert Templates

About this task

To create and use a custom exception template:

Procedure

1. Copy the *install_dir*/repository/xapi/template/merged/exception_console/example_exception_console.xsl template file.
2. Modify the file as needed, rename it, and save it within the *install_dir*/extensions/global/template/xsl/*CUSTOM-TEMPLATE-XSL*

directory. You can save it to another directory, but using the standard directory structure supplied by the application helps ensure consistency.

3. From the Service Builder, configure an Alert node to use your custom exception console template. In the XSL Template field, specify template/xsl/*CUSTOM-TEMPLATE-XSL*.

   Note: Using the *install_dir*/repository/xapi/template/merged/exception_console/*file_name*.ect file with Actions to accomplish the same purpose has been deprecated in Release 5.0. Use the Alert node and an XSL file instead.

Publish Event Handler

The application enables publishing of data by the event manager to external systems for interoperability. You can configure any event to publish data to other systems. For example, upon the event ON_SUCCESS of the SHIP_CONFIRM transaction, you may want to publish data to an external financial system.

When configuring the Publish event handler, you must provide a set of system IDs separated by semicolons (for example, SYSTEM1;System2;TestSystem). These are IDs of time-triggered Transactions that are expected to read and process the data. The event manager writes the data provided by the transaction to the YFS_EXPORT table and writes one record for each system ID configured.

For publishing data using a service, use a Database transport node. For more information, see the Sterling Selling and Fulfillment Foundation : *Configuration Guide*.

Execute Event Handler

The Execute event handler tells the event manager to invoke the specified program. The application looks for this program in the PATH. Data is passed to the program as the first command line argument. Like the Publish event handler, the Execute event handler is invoked asynchronously. This is achieved by using a custom API that runs the program in a service.

Java Extensions as Event Handlers

Java extensions enable you to implement event handlers as Java classes. With each action, you can associate one Java event handler. The Java class you create must implement the com.yantra.yfs.japi.util.YFSEventHandlerEx interface. This is achieved by using a custom API which actually runs the program in a service.

Oracle Stored Procedures as Event Handlers

If you configure an event handler as an Oracle stored procedure, the stored procedure is invoked with the following parameters:

```
PROCEDURE DO_ACTION(TRANID IN VARCHAR2,ACTIONCODE IN VARCHAR2,KEY_DATA IN
VARCHAR2,DATA_TYPE IN NUMBER,DATA_BUFFER1 IN VARCHAR2,DATA_BUFFER2 IN
VARCHAR2,DATA_BUFFER3 IN VARCHAR2,DATA_BUFFER4 IN VARCHAR2,DATA_BUFFER5 IN
VARCHAR2,DATA_COMPLETE IN NUMBER,SHIP_NODE IN VARCHAR2,RETURN_VALUE IN OUT NUMBER)
```

PL/SQL limits the maximum size of a VARCHAR2 variable to 2,000 bytes, so 2,000 bytes is the maximum length of the string passed in data buffers. If the input is greater than 10,000 characters, the buffer is truncated and the parameter

DATA_COMPLETE has a value of 0 (zero). If the buffer is less than 10,000 bytes and is completely passed to the stored procedure, DATA_COMPLETE is passed as 1.

Important: You must not do a commit or a rollback in the stored procedure at any time in a stored procedure associated to a event handler.

Execution of stored procedures is not a supported service component.

HTTP Extension as Event Handlers

If you configure an event handler as an HTTP extension, the application sends data using the post() function to the specified URL. Data is posted using the variables in the following table.

Variable	Description
sTranID	ID of the transaction raising the event.
iDataType	If iDataType is set to 1, XML data is published with the event and sData contains the entire XML string. Otherwise, iData Type is set to 0 and sData contains a name=value pair. To see what the name=value pair contains, see the DBD file associated with the event in the Javadocs.
sData	See iDataType description.
sShipNode	Ship node ID, if available.

A service can be configured to post data using the HTTP protocol. For details, see the Sterling Selling and Fulfillment Foundation *Configuration Guide*.

COM Extensions as Event Handlers

Using COM requires setting up your server and runtime clients.

If you define an event handler as a COM extension, the application calls the COM component you specify as the COM object. The COM component you create must expose the IDispatch interface and a pre-identified function doAction() as part of the interface. The signature of the doAction() function should be as follows:

```
[IDL Syntax]
doAction ( [in] BSTR sTranID, [in] BSTR sActionCode, [in] BSTR sKeyData, [in]long
DataType, [in] BSTR sData, [in] BSTR ShipNode, [Out] long *Retval);
```

Because the application accesses this function through the IDispatch interface, you can write your COM component either in the Visual Basic or C++ programming language. The function signature that you must expose through Visual Basic is:

```
Public Function doAction(ByVal bsTranID As String, ByVal boActionCode As String,
ByVal bsKeyData As String, ByVal DataType As Long, ByVal bsData As String, ByVal
ShipNode As String, retVal as Long) As Long
```

The function signature that you must expose through C++ is:

```
STDMETHODIMP COMActionImpl::doAction(BSTR sTranID, BSTR sActionCode, BSTR sKeyData,
long DataType, BSTR sData, BSTR Node, long *RetVal)
```

COMActionImpl is the class name that implements the doAction() function. While configuring the event handler, you must specify the Program ID of the COM component to be invoked.

Event Chaining and Event Handlers

Some event handlers support invocation of APIs. These APIs can retrieve more data pertinent to the event being raised, or they may direct to perform a business computation or transaction. In turn, the transaction can raise other events, thus resulting in event chaining. For example, one of the event handlers for the ORDER_CREATE event can check the validity of an order and call an API to HOLD the order. This results in the triggering of the HOLD event, which sends an e-mail message to a customer service representative. Event chaining provides an extensibility mechanism that is required for complex business processing.

Important: Actions that are configured to be invoked by the ON_FAILURE event must not perform any database updates. Any database updates performed by an action invoked by the ON_FAILURE event is rolled back.

To access the data published by Sterling Selling and Fulfillment Foundation , you must implement the YFSEventHandlerEx() interface, which provides the handleEvent() function to implement as follows:

```
public boolean handleEvent (YFSEnvironment oEnv, String sTranID, String
 sActionCode, Map sKeyData, int iDataType, Object sData, String sShipNode,
 String [] parms) throws Exception
```

Important: While it is possible to implement an API that causes an event that is associated with an action in the application to call the same API, take care to avoid this situation. It creates an infinite loop.

The class name must be configured as the Java object on the Action Configuration screen in theSterling Selling and Fulfillment Foundation transaction configuration. For more information, see the Sterling Selling and Fulfillment Foundation *Configuration Guide*.

All the parameters for the handleEvent() function are input parameters, with values inserted by the application. The last parameter in this function is the parms parameter, which is an array of String constants. The values of these string constants are the values of the string constants specified after the class name during the action configuration in the application.

Note: For details on the YFSEventHandlerEx interface, see the Javadocs.

Chapter 3. Using Variables for Configuration

Using Variables for Dynamic Configuration

You can dynamically configure certain properties belonging to actions, agents and services configuration to cut back on your implementation time. For example, you can provide a dynamic way of configuring network properties such as provider URLs, E-mail server, and Sender addresses. These properties can be configured as variables in one place and can be resolved at runtime when these properties are being used.

You can provide variables in place of file or directory names wherever a file name or path can be entered in the Applications Manager. This variable substitution is based on an entry in the *install_dir*/properties/customer_overrides.properties file and is resolved during runtime.

You can use variable names in the following components of the Applications Manager:

- Service Definition Framework
  - All transport types such as JMS queue name, Initial Context Factory, QCF lookup and Provider URL. File Sender and Receiver, FTP source and destination for sender and receiver directories and HTTP and Webservice transport types.
  - In the e-mail component, where you can specify, the email server, subject, listener port, and From addresses. The dynamic configuration can also be used for specifying the email protocol.
  - Actions
    - In call HTTP extension and Execute Program.
  - Agent criteria details
    - JMS Queue Name, Initial Context Factory, QCF Lookup and Provider URL.
- Printer Devices, Print Documents, and Print Components
- Purge Criteria's log file name
- In the System Management console:
  - You can use variables to specify installation rules such as e-mail server name, server IP address, server listener port, and e-mail protocol.
  - You can provide variables for the JMS monitoring configuration fields, which include: WebLogic Provider URL, WebSphere Channel name, host name, port number and queue manager name.

For the fields identified above, you can configure the values as yfs.VARIABLE_NAME in the *install_dir*/properties/customer_overrides.properties file. It is stored in the database as is and at runtime when the variables are used, a lookup is performed in the customer_overrides.properties file to decipher the value. Since the values for these variables are fetched from the customer_overrides.properties file, they are specific to a particular JVM.

Note: The value of this variable cannot be seen in the health monitor agent details, since the value depends on the JVM on which it is deployed. You have to click on the server details of the monitor agent to view the value of the variable.

For example, if you want to set the File IO Receiver's directory structure to a common variable (such as ${ffbase}) , then the incoming directory should be set to:

```
${ffbase}/incoming
```

The value for the variable ${ffbase} that you defined in the Applications Manager must be defined in the customer_overrides.properties file with a prefix of yfs as:

```
yfs.ffbase=C:/FileIODir/Receiver
```

This ${ffbase}/incoming value is stored in the database, and when processing the file adapter, the variable is resolved to C:/FileIODir/Receiver/incoming.

The following conditions are assumed for the usage of this variable:

- All the variables when referenced must be in the following format:
  - ${variable_name}
- All variables should be properly formed. If a variable is not found, no substitution takes place.
- Variables must not contain the '}' character.
- Variables must not begin or end with a whitespace character.
- Templates do not support variables for filenames since they are always resolved within the classpath.

Chapter 4. Creating Custom Time-Triggered Transactions

About Custom Time-Triggered Transactions

The application provides infrastructure that enables you to write your own time-triggered transactions. You invoke and schedule these time-triggered transactions in much the same manner as you invoke and schedule standard time-triggered transactions.

For information on how to configure the standard time-triggered transactions, see the Sterling Selling and Fulfillment Foundation *Configuration Guide*.

Depending upon the way time-triggered transactions determine the list of tasks to be processed (their work load), they can be classified into one of the following categories:

- Non task-based (generic) - these time-triggered transactions use custom logic to determine the work they have to perform. They may or may not use the centralized Task Queue.
- Task-based (specific) - these time-triggered transactions use the Task Queue to determine their work.

The application provides infrastructure to create both types of custom time-triggered transactions.

The ability to write non-task-based time-triggered transactions is provided using the com.yantra.ycp.japi.util.YCPBaseAgent class. This class provides a generic infrastructure irrespective of whether the Task Queue is used or not and therefore can be used to write any time-triggered transaction.

Task-based time-triggered transactions can be programmed by subclassing the com.yantra.ycp.japi.util.YCPBaseTaskAgent class.

If your transaction is Task Queue based, it is suggested that you use the infrastructure provided specifically to write task-based transactions. This infrastructure automatically determines work for your custom agent from the Task Queue, thus reducing the amount of design and development required for your transaction.

All the custom agents written to the application specification are subclassed from the com.yantra.ycp.japi.util.YCPBaseAgent class, which has two abstract functions. When you implement these functions, they provide the processing capabilities of a time-triggered transaction.

getJobs() Abstract Function

The getJobs()abstract function uses:

- Env, a pre-created instance of a YFSEnvironment object that can be passed to APIs
- inXML, an org.w3c.dom.Document object which contains the input XML

The implementation of this function should obtain jobs (from the database) that need to be run, construct a list of org.w3c.dom.Document objects, and return the list. See the following signature:

```
public List getJobs(YFSEnvironment Env, Document inXML)
```

executeJobs() Abstract Function

Each document in the List returned by the getJobs() function is passed to this function for execution. If this function throws an exception, the exception is logged and the transaction is rolled back, otherwise it is committed. See the following signature:

```
public Document executeJob(YFSEnvironment Env, Document inXML)
```

After all of the documents have been processed by the executeJob() function, the application invokes the getJobs() function again to obtain the next set of tasks that need to be processed by the executeJob() function. This repeats until no more jobs are returned by the getJobs() function.

For examples of the input XML, see the YCPBaseTaskAgent class in the Javadocs.

The com.yantra.ycp.japi.util.YCPBaseAgent also provides utility functions for trace logging and timer information with the following signatures:

- public void log(String classMethodName, String message);
- public startTimer(String timerName);
- public endTimer(String timerName);

Creating Custom Non-Task-Based, Time-Triggered Transactions

About this task

Custom non-task-based time-triggered transactions should be written as subclasses of the com.yantra.ycp.japi.util.YCPBaseAgent class. For a sample custom time-triggered transaction, see the com.yantra.ycp.japi.util.YCPBaseAgent class in the Javadocs.

To write a custom non-task-based time-triggered transaction:

Procedure

1. Subclass com.yantra.ycp.japi.util.YCPBaseAgent.
2. Implement the executeJob() and getJobs() functions in this class.
3. From the Applications Manager, configure a time-triggered transaction and assign an Agent Server to it.
4. Schedule and run your custom time-triggered transaction according to the instructions in the Sterling Selling and Fulfillment Foundation: *Installation Guide*.

About Creating Custom Task-Based, Time-Triggered Transactions

Task-based custom time-triggered transactions are written as subclasses of the com.yantra.ycp.japi.util.YCPBaseTaskAgent class, which is a subclass of com.yantra.ycp.japi.util.YCPBaseAgent with the getJobs() and executeJob() functions already implemented. Creating a task-based custom time-triggered

transaction by subclassing this class involves implementing the executeTask() function to process one task queue record passed to you as input.

Note: If the executeTask() function throws an exception, the exception is logged and the transaction is rolled back, otherwise it is committed.

The logging and timing utility functions available are similar to the ones provided by the com.yantra.ycp.japi.util.YCPBaseAgent class. The signature of the executeTask() function is public Document executeTask(YFSEnvironment oEnv, Document inXML);Env is a pre-created instance of a YFSEnvironment object that can be passed to APIs and InXML is the org.w3c.dom.Document object, which contains the custom task XML. The custom task XML also contains a TransactionFilters Node, which contains all the parameters passed to the task-based custom time-triggered transaction. This node is below the root node of the input XML. For example, see an example for a task-based custom time-triggered transaction.

```
<?xml version="1.0" encoding="UTF-8"?>
<TaskQueue  TaskQKey="" TransactionKey="" DataKey="" DataType="" AvailableDate=""
     Lockid="" Createts="" Createprogid="" Createuserid=""
     Modifyts=""Modifyprogid="" Modifyuserid="" >
                 <TransactionFilters   AgentName=""  TransactionKey=""
               CurrentThread="" NumRecordsToBuffer="" TotalThreads=""/>
</TaskQueue>
```

For a sample custom task-based time-triggered transaction, see the com.yantra.ycp.japi.util.YCPBaseTaskAgent class in the Javadocs.

Creating Custom Task-Based, Time-Triggered Transactions

About this task

To write a task-based custom time-triggered transaction:

Procedure

1. Subclass com.yantra.ycp.japi.util.YCPBaseTaskAgent.
2. Implement the executeTask() function in this class.
3. From the Applications Manager, configure a time-triggered transaction and assign an Agent Server to it.
4. Schedule and run your custom time-triggered transaction according to the instructions in the Sterling Selling and Fulfillment Foundation: *Installation Guide*.

Chapter 5. External Transactions

Coordinating with External Transactions

The application provides the capability to run external code either by raising an event, calling a user exit, or invoking a custom API or service. During these invocations, the external systems can begin a transaction. Since the external transaction is not part of the Sterling Selling and Fulfillment Foundation transaction it could lead to data inconsistencies if the Sterling Selling and Fulfillment Foundation transaction is rolled back but the external transaction is not.

External transaction coordination lets the external systems register their transactions with Sterling Selling and Fulfillment Foundation. When Sterling Selling and Fulfillment Foundation is ready to commit its transaction, the YFSTxnCoordinatorUE user exit is invoked, which lets the application handle the commits and rollbacks on the external transactions.

Implement External Transaction Coordination

About this task

To implement external transaction coordination:

Procedure

1. Create the custom API that you want to implement your external transaction process, but DO NOT commit the external transactions. Instead, register the external transaction object with the YFSEnvironment by calling the (YFSEnvironment)env.setTxnObject(String ID, Object txnObj).

   The String ID is a unique name used by the user exit implementation class to identify the custom transaction object.

   The following example illustrates a simple custom API.

```
public class doSomethingAPI
{
private YFCLogCategory cat = YFCLogCategory.instance("DoSomethingAPI");
public doSomethingAPI()  // constructor is empty
{
}
public void writeToDB (String key, YFSEnvironment oEnv)
{
try
{
Driver aDriver =
(Driver)Class.forName("oracle.jdbc.OracleDriver").newInstance();
        String url = "jdbc:oracle:thin:@127.0.0.1:1521:qotree2";
        Connection conn = DriverManager.getConnection(url, "Scott", "Tiger");
conn.setAutoCommit(false);
       String sql = "insert into TxnTest (key) values ('" + key + "')";
       Statement stmt = conn.createStatement();
       stmt.executeUpdate(sql);
       oEnv.setTxnObject("YDBconn", conn);
}
catch (Exception e)
{
System.out.println ("Caught Exception :\n" + e);
}
   }
```

```
public Document doSomething(YFSEnvironment env, Document doc) throws Exception
{
      System.out.println("Executing doSomething method..........");
writeToDB ("doSomething", env);
return doc;
   }
}
```

2. Implement the com.yantra.yfs.japi.ue.YFSTTxnCoordinatorUE user exit interface to commit the external transactions either before or after Sterling Selling and Fulfillment Foundation perform its commits. Then implement the rollback method so that if the Sterling Selling and Fulfillment Foundation transaction triggers a rollback, the afterYantraTxnRollback(YFSEnvironment oEnv) method is called to rollback the external transactions as well. Implement the following methods to accomplish this:
   - beforeYantraTxnCommit(YFSEnvironment oEnv)
   - afterYantraTxnCommit(YFSEnvironment oEnv)
   - afterYantraTxnRollback(YFSEnvironment oEnv)

   Note: You can use either the beforeYantraTxnCommit or the afterYantraTxnCommit user exit to synchronize commits, depending on your integration requirements.

   Calling(YFSEnvironment)env.getTxnObject(ID)enables these methods to obtain the handle to the external transaction object that was previously registered by the (YFSEnvironment)env.setTxnObject(String ID, Object txnObj). Note the ID is the same in both the getTxnObject call and the setTxnObject call.

   The following is an example of the YFSTTxnCoordinatorUE user exit interface implementation.

```
public class afterTxnCommit implements YFSTxnCoordinatorUE
{

public void beforeYantraTxnCommit (YFSEnvironment oEnv) throws YFSUserExitException
{
// before method is not implemented because after method is implemented.
}
public void afterYantraTxnCommit (YFSEnvironment oEnv) throws YFSUserExitException
{
System.out.println ("Entering method afterYantraTxnCommit.........");
try
{
   Connection ydbConn = (Connection)oEnv.getTxnObject("YDBconn");
   ydbConn.commit();
}
catch (Exception e)
{
System.out.println ("Caught Exception :\n" + e);
}
}
public void afterYantraTxnRollback (YFSEnvironment oEnv) throws
YFSUserExitException
{
System.out.println ("Entering method afterYantraTxnRollback.........");
try
{
   Connection ydbConn = (Connection)oEnv.getTxnObject("YDBconn");
   ydbConn.rollback();
}
catch (Exception e)
{
System.out.println ("Caught Exception :\n" + e);
}
}
```

3. Launch the Applications Manager and navigate to **System Administration > User Exit Management** to configure the YFSTxnCoordinatorUE Implementation class.

   For more information about the YFSTxnCoordinatorUE user exit interface definition, see the Javadocs.

Chapter 6. Transactional Data Security

How Is Transactional Data Secured?

Security issues involve controlling who has access to what data, how much they can see, and what they can do with it. The application provides mechanisms that address the following security issues:

- User access control
- Single sign on
- Data encryption

User Access Control

Through an access control mechanism of user group permissions, logins, and order modification permissions, the application provides security measures for multiple levels of customer service and administrative organizations.

APIs control access to different areas of system functionality. It is the responsibility of the caller to ensure that the invoking user has access rights for the function being invoked. The application provides security manager APIs to help in this effort. See the com.yantra.api.ycp.security package in the Javadocs.

Single Sign On

If a single sign on mechanism is required within the application, implement the interface com.yantra.ycp.japi.util.YCPSSOManager, which has a getUserData() function that returns a String(UserId). For more detailed information, see the Javadocs.

Data Encryption

Encryption ensures that sensitive data is not viewed by unauthorized people. The application provides APIs that enable you to encrypt data such as user names, passwords, and credit card numbers.

In addition, encryption and decryption is only applied after it has been specified within the Applications Manager. For example, only user exits that have been passed credit card information can access decrypted credit card numbers.

Encryption Logic

The application exposes the com.yantra.ycp.japi.util.YCPEncrypter interface to handle encryption logic. All application encryption and decryption is handled by an encrypter class that implements this interface. This class is specified by configuring the following properties in the *install_dir*/properties/customer_overrides.properties file:

- yfs.encrypter.class
- yfs.propertyencrypter.class properties

Both classes must implement the com.yantra.ycp.japi.util.YCPEncrypter interface.

The com.yantra.ycp.japi.util.YCPEncrypter interface has the following two functions:

- public java.lang.String encrypt(java.lang.String sData) - sData is the data passed by the application to the implementing class for encryption. The return value is the encrypted string.
- public java.lang.String decrypt(java.lang.String sData) - sData is the data which is required to be decrypted.

For information on writing your own property encrypter class, see the YCPEncrypter interface in the Javadocs.

Encryption and decryption functions in this interface are invoked multiple times by Sterling Selling and Fulfillment Foundation . The application does not distinguish between clear text and encrypted information. Therefore, the encrypt function may be invoked with previously encrypted data. In order to avoid double encryption, it is important for the encrypt function to be able to distinguish between clear text and previously encrypted information. If previously encrypted information is passed to the function, your implementation of this function should return what is passed into it without encrypting it again.

The decrypt function also should be able to distinguish between clear text and previously encrypted text.

Disabling Encryption and Decryption

To disable encryption (or decryption), implement the encrypt (or decrypt) function to return the same value it is passed as input without any processing.

Choosing an Encryption and Decryption Strategy

There are multiple deployment options when choosing an encryption strategy. The most typical options are:

- No encryption or decryption
- External tokenization
- Both encryption and decryption
- No decryption

Note: It is recommended that you use external tokenization as your encryption strategy.

Use the following explanation to guide your decision-making process:

Using No Encryption and No Decryption

Do not save clear text credit cards numbers in the database. However, you may configure encryption in Sterling Selling and Fulfillment Foundation , if:

- Your business does not accept, process, or store credit card numbers or other sensitive information.
- All encryption and decryption is handled externally. The application passes the externally encrypted credit card numbers to other systems. If you enable encryption, strings that are encrypted would be encrypted again.

Note: Storing encrypted PANs, even if encryption and decryption is not implemented, may put the application into PCI DSS and PA-DSS auditing scope.

External Tokenization

TheSterling Sensitive Data Capture Server application captures and tokenizes credit card numbers and store value card numbers. It is recommended that you review the Sterling Selling and Fulfillment Foundation : *Secure Deployment Guide* for the approach to meeting PCI DSS and PA-DSS requirements.

Using Both Encryption and Decryption

The application encrypts and decrypts credit card numbers automatically as required. However, this strategy is not the recommended encryption strategy.

Using Encryption But No Decryption

If your business requiresSterling Selling and Fulfillment Foundation to store credit card numbers, but you never want Sterling Selling and Fulfillment Foundation to automatically decrypt them under any circumstances, you may want to enable only the encrypt function and disable the decrypt function.

This way,Sterling Selling and Fulfillment Foundation encrypts the credit card numbers passed in as clear text but never converts them back. OnceSterling Selling and Fulfillment Foundation encrypts the information, all your custom extensions are passed as encrypted credit card numbers and must handle decryption externally. It is important to note that a few user exits inSterling Selling and Fulfillment Foundation (for example, YFSbeforeCreateOrderUE) are invoked before the credit card number is encrypted, so it still has access to the clear text number.

How is Encryption Supported?

The application supports encryption for the following places:

- Properties specified in the yfs.properties, yifclient.properties, and management.properties files
- Credit card numbers

Encryption Through yfs.properties

Properties such as the JDBC URL, database User ID and Password can be stored encrypted in the customer_overrides.properties file. Because the application needs this information to connect to the database, these values must be decrypted by the application. If you do not wish the application to ever decrypt data, these properties cannot be stored encrypted.

Note: If you want set any of the properties specified in the yfs.properties file, add an entry for that particular property in the *install_dir*/properties/customer_overrides.properties file.

Encryption for Credit Card Numbers

The application can encrypt credit card numbers before storing them in the database. Unlike the properties specified in the yfs.properties file, decrypted credit card numbers are never required by the application for default processing. However, you may extend the application by implementing a user exit that requires decrypted credit card numbers for charging or storing user preferences. If you don't want the application to decrypt information automatically, you must decrypt these credit card numbers in your implementation of the user exit.

Encrypting Credit Card Numbers

If you implement encryption, the application encrypts credit card number in the following situations:

- When returned by an API
- When published as a part of event data
- When stored anywhere in the database
- When displayed on the user interface (although the user interface may have an option to override this behavior based on user access)

Encrypting Credit Card Numbers Through APIs

Note: It is recommended that payment information entering the system be already tokenized instead of being encrypted.

The application provides the following APIs that enable you to encrypt and decrypt credit card numbers assuming that both encryption and decryption algorithms have been implemented by the Encrypter class:

- getEncryptedString() - accepts a string passed to it and returns the string encrypted
- getDecryptedString() - accepts an encrypted string and returns the string decrypted
- getEncryptedCreditCardNumber() - returns an encrypted credit card number

  Note: getEncryptedCreditCardNumber() has been deprecated in Release 9.0. It has been replaced with getEncryptedString().
- getDecryptedCreditCardNumber() - returns the credit card number that had been encrypted using the getEncryptedCreditCardNumber() API

Note: getDecryptedCreditCardNumber() has been deprecated in Release 9.0. It has been replaced with getDecryptedString().

Encrypting Credit Card Numbers Through User Exits

Only user exits that are passed credit card information can access decrypted credit card numbers. The application provides the following user exits for passing credit card data:

- YFSCollectionCreditCardUE
- YFSCollectionCustomerAccountUE
- YFSCollectionOthersUE
- YFSCollectionStoredValueCardUE
- YFSValidateInvokedCollectionUE

Note: Some user exits, such as YFSBeforeCreateOrder and YFSBeforeChangeOrder, may have access to credit card numbers before they are encrypted in the system.

For detailed information about these user exits, see the Javadocs.

Encryption Through Property Files

About this task

Some properties relay sensitive data such as user IDs and passwords, which you may want to encrypt. Any property (except for the yfs.propertyencrypter.class property in the yfs.properties file) mentioned in the following files can be encrypted using the *install_dir*/properties/customer_overrides.properties file:

- *install_dir*/properties/yfs.properties
- *install_dir*/resources/yifclient.properties files

To encrypt properties:

Procedure

1. Prefix the property value you want to encrypt with `encrypted:`. For example,
   `yfs.dblogin.datasource.name=encrypted:`*`encrypted value`*
2. Ensure that the security.propertyencrypter.class property is accessible through the CLASSPATH environment variable.
3. Implement the YCPEncrypter interface. For details about this interface, see the Javadocs.

   These properties starting with `encrypted:` are automatically decrypted at runtime.

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.

1623-14, Shimotsuruma, Yamato-shi

Kanagawa 242-8502 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 2011. Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. 2011.

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