

Sterling Selling and Fulfillment Foundation



Customization Basics

Version 9.1

Sterling Selling and Fulfillment Foundation



Customization Basics

Version 9.1

Note

Before using this information and the product it supports, read the information in "Notices" on page 51.

Copyright

This edition applies to the 9.1 Version of IBM Sterling Selling and Fulfillment Foundation and to all subsequent releases and modifications until otherwise indicated in new editions.

© **Copyright IBM Corporation 1999, 2011.**

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

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	3
Build and Deploy your Customizations or Extensions	3

Chapter 2. Extensibility Overview 5

Extending Your Application	5
Extending the Console User Interface	5
Extending the Applications Manager User Interface	6
Extending the Mobile User Interface	6
Extending the Database	7
Extending Transactions	7
Extending the Rich Client Platform User Interface	8

Chapter 3. Setting Up the Development Environment 9

Prerequisites for Extending Your Application	9
Understanding the Development Environment	9
Preparing the Development Environment on WebLogic	9
Preparing the Development Environment on WebSphere	12
Preparing the Development Environment on JBoss	14
Developing and Testing in the Development Environment	15
Testing UI Customizations	16
Configuring the UI Cache Refresh Actions	16
Configure Resource Cache Refresh Actions	16

Chapter 4. Customization Using Microsoft COM+ 17

Microsoft COM+ Prerequisites	17
Creating a COM+ Application on Windows	17
Adding Components to a COM+ Application	17
Configuring the COM+ Service	18
Creating a Client Proxy	19
Installing a Client Proxy	19

Chapter 5. Masking Sensitive Information During Logging 21

Masking Sensitive Information During Logging Using Log4j	21
--	----

Chapter 6. Data Validation 23

About Data Validation	23
Disabling Data Validation	24
Bypassing Data Validation for an URI	24

Implementing Data Validation	25
Defining Regular Expressions in Datatypes XML File	25
Defining Regular Expressions in XML Files	25
Registering Regular Expressions	27
Defining Validation Rules in Datatypes XML File	28
Externalizing Validation Rules Defined in the Datatypes XML File	28
Defining Validation Rules in XML Files	29
Defining Abstract Validation Rules	31
Extending Abstract Validation Rules	32
Registering Validation Rules	32
Overriding Regular Expressions	33
Overriding Validation Rules	33
Defining an Adapter to Find Validation Rules	34
Defining URI-Based Adapter to Find Validation Rules	34
Deleting Registered Validation Rules	34
Exception Handling	35
Defining Error Messages	36
Localizing Error Messages	36
Defining Custom Regular Expression Error Message Provider	36
Localizing Validation Rules	37

Chapter 7. Building and Deploying Extensions 39

After You Create Your Extensions	39
Building Resource Extensions	39
Building Other Extensions	40
Building Database Extensions	41
Deploying Extensions	41
Building and Deploying Enterprise-Level Extensions	42
Building Enterprise-Level Extensions	42
Building Enterprise-Level Resources Extensions	43
Building Enterprise-Level Database Extensions	43
Building Enterprise-Level Template Extensions	44
Deploying Enterprise-Level Extensions	44
Customizing Web.xml	44
Customizing web.xml for Multiple Applications	44
Customizing web.xml for Session Timeouts	45
Deploying the Enterprise Archive Package	45
Deploying Multiple EARs on One Application Server	46
Defining the JNDI Context Namespace	46
Defining Context Root Entries	47

Chapter 8. File Names, Keywords, and Other Conventions 49

Reserved Special Characters and Keywords Introduction	49
Naming Files	49
Reserved Keywords	49
Using Multi-Byte Characters	50

Notices 51

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. Extensibility Overview

Extending Your Application

This chapter introduces the types of extensibility possible through Sterling Selling and Fulfillment Foundation. It provides an overview, describes high-level concepts, and provides technical architectural diagrams of the application.

- Look and Feel Extensibility

The application provides a Presentation Framework toolkit that enables you to change the way information is rendered or displayed without changing the way it functions.

- Transactional Extensibility

Sterling Selling and Fulfillment Foundation provides a Service Definition Framework, which is an infrastructure that automates the conversion and transportation of data between Sterling Selling and Fulfillment Foundation and third-party applications, and then converts that data into formats readable by each system. The Service Definition Framework also handles logging and exceptions. It enables you to create custom transactions that extend the functionality of the application.

- Database Extensibility

In addition to customizing the user interface and transactions, you can extend the database to store additional attributes specific for your business.

- Printed Documents Extensibility

You can customize printed documents. For example, you can extend the default length of bar codes.

You can use JasperReports for generating printed reports in the application. Additionally, you can also generate reports as a PDF, RTF or any other Jasper Print object in the application. JasperReports is an open source Java reporting tool, which you download and install separately. The JasperReports installation guidelines are provided in the *INSTALL_DIR/xapidocs/code_examples/jasperreports/alert_report_readme.html* file. A sample JasperReport called *sampleAlertReport.pdf* is also available in the same directory.

A Jasper print component is defined in the Service Definition Framework which can be used to automatically print a document based on an event. It is a standard XML-based component that accepts XML as input and provides an output XML. For more information on this component and the print transaction, refer to the Sterling Selling and Fulfillment Foundation: *Application Platform Configuration Guide*.

- Browser Portal Extensibility

You can export views into the application database that lists a user's frequently used search criteria.

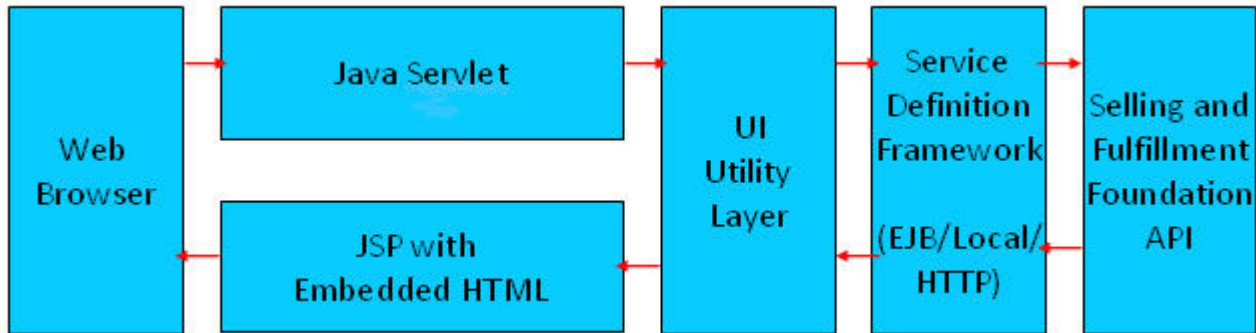
Extending the Console User Interface

The Application Console is the user interface for conducting and tracking day-to-day transactional business, such as orders and inventory.

The Application Console UI uses HTML within Java Server Pages (JSPs). The user interface layer accesses the exposed APIs through services defined in the Service Definition Framework, which ensures that only exposed APIs are used.

The UI layer of the Service Definition Framework uses very minimal XML manipulation. Wherever significant manipulation of XML output becomes necessary, changes to the APIs provide a more UI friendly output.

The following figure shows the technical architecture of the Application Console user interface.



For detailed information about extending the Application Console user interface, see the Sterling Selling and Fulfillment Foundation: *Customizing Console JSP Interface for End-User Guide*.

Extending the Applications Manager User Interface

The Applications Manager is the user interface for configuring the setup of an organization's business rules and transactions. It is composed of Java Swing pages.

For detailed information about extending the user interface, see Sterling Selling and Fulfillment Foundation: *Customizing the Swing Interface Guide*.

Extending the Mobile User Interface

Sterling Selling and Fulfillment Foundation enables you to develop and display a custom user interface for mobile devices used in warehouse operations.

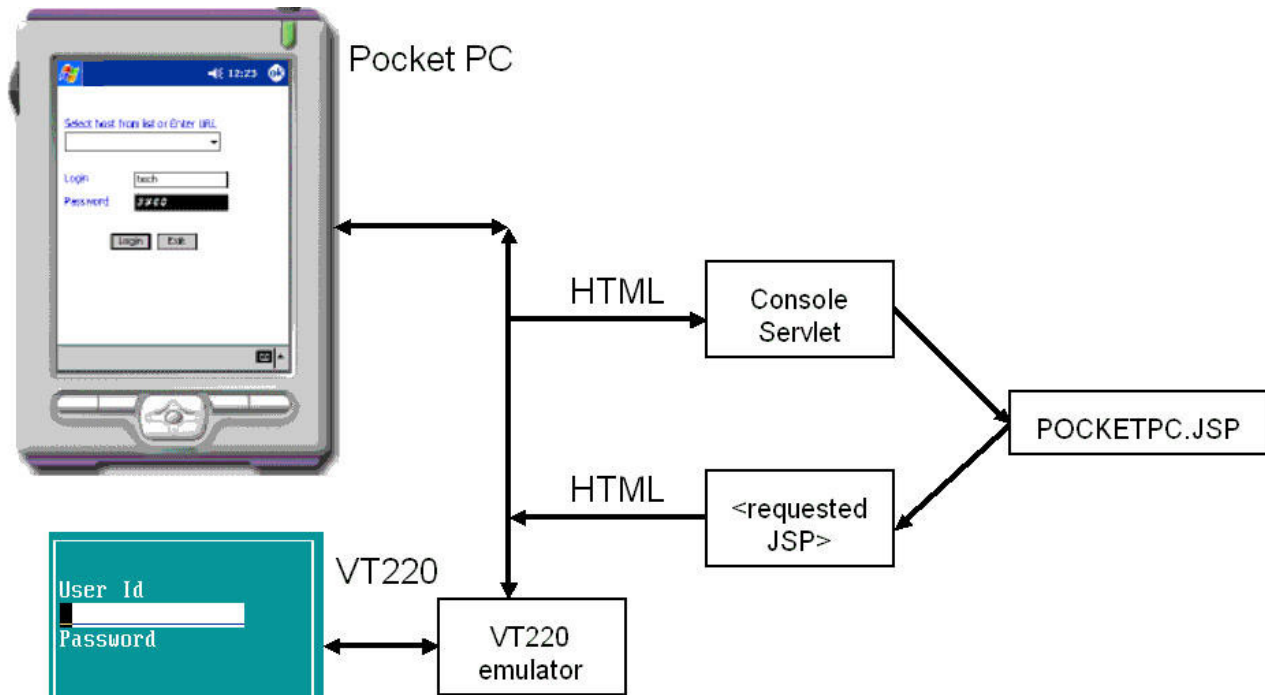
The mobile architecture consists of two components: A client and a server.

- Mobile client—Typically a Pocket PC-based handheld device, but it can also be a VT220 emulation terminal. In this documentation, the mobile client is also referred to as a mobile device.
- Application server— Sterling Selling and Fulfillment Foundation running on an application server.

The client and server communicate using the HTTP protocol to transfer HTML according to the request and response model. Each client request is of the type `"/console/*.ppc"`.

The application server directs requests for any.ppc to the controlling servlet (called the Console Servlet) which in turn redirects the request to the pocketpc.jsp file. The pocketpc.jsp file redirects the request to the JSP as specified in the uidentity. The JSP renders an HTML response, which is displayed by the mobile client.

The following figure illustrates this architecture.



For detailed information about extending the mobile user interface, see Sterling Selling and Fulfillment Foundation : *Customizing User Interfaces for Mobile Devices Guide*.

Extending the Database

Database extensibility enables you to add columns and tables to capture additional data.

For more information, see Sterling Selling and Fulfillment Foundation: *Extending the Database Guide*.

Extending Transactions

Sterling Selling and Fulfillment Foundation provides a mechanism for processing and resolving errors during data transformation and transportation. This mechanism, called the Service Definition Framework, enables access to the following transactional processes:

- System APIs exposed by the application
- Event handlers to route the data published by the application to the transport services layer
- Time-triggered transactions that monitor and run tasks as needed

The Service Definition Framework provides error checking through the log4j utility, which writes both trace and debug information to a log file.

In addition, you can extend the application by creating the following types of custom code:

- Extended (custom) APIs
- User exits that override the default business algorithm

- User exits that extend the business algorithm used by APIs and time-triggered transactions
- Custom time-triggered transactions

For more information about extending transactions, see Sterling Selling and Fulfillment Foundation : *Extending Transactions Guide*.

Extending the Rich Client Platform User Interface

The Rich Client Platform provides a highly interactive Rich Client that can be remotely deployed, updated, and easily managed. A Rich Client is a client that processes the bulk of data operations without depending on the server to which it is connected. However, it is dependent on the server, primarily for data storage. The Rich client is rich in features and functionality and has complete access to the programming functions of the operating system.

For more information, see Sterling Selling and Fulfillment Foundation : *Customizing the Rich Client Platform Interface Guide*.

Chapter 3. Setting Up the Development Environment

Prerequisites for Extending Your Application

This guide assumes that you:

- Have already installed the application.
- Are familiar with creating and running the Enterprise Archive (EAR) file in deployment mode, as described in the Sterling Selling and Fulfillment Foundation: *Installation Guide*.
- Are familiar with the standard (factory default) installation.

Throughout this guide, *INSTALL_DIR* refers to the directory where you have installed Sterling Selling and Fulfillment Foundation.

Understanding the Development Environment

The development environment you need depends on the type of work you are doing.

If you intend to customize Sterling Selling and Fulfillment Foundation, you need a test environment that enables you to verify that your changes work as you intend. To save development time, you can customize your test environment to run the application in development mode.

If you extend the database, include the *yfsdbextn.jar* file before the *yantrashared.jar* in the CLASSPATH in all scripts.

If you have installed any Packaged Composite Application (PCA), for example, Sterling Call Center and Sterling Store, the *yantrautil.jar* should be removed from the application server CLASSPATH before starting the application server to run the application in development mode.

Development mode saves time by enabling your application server to automatically load the latest version of edited JSP files directly from specific directories rather than reading them from the Sterling Selling and Fulfillment Foundation EAR file. This enables you to customize and test iteratively, without having to repeatedly create the EAR file.

Development mode also enables you to immediately test UI customizations.

The way you set up the development environment depends on the application server you use.

Preparing the Development Environment on WebLogic

About this task

To enable WebLogic to run Sterling Selling and Fulfillment Foundation without creating an EAR, you must define an application in WebLogic with the appropriate settings and then configure your startup script to set up the CLASSPATH required by the application.

Setting up the application directory structure enables WebLogic to read from files directly rather than from the EAR file.

To configure WebLogic to run application in exploded mode:

Procedure

1. Edit the `<WEBLOGIC_DOMAIN>/bin/startWebLogic.cmd` script for windows (`startWebLogic.sh` for UNIX), and set the following argument in Options as Java parameters:

```
-Dsci.opsproxy.disable=Y -Dvendor=shell  
-DvendorFile=/servers.properties
```
2. Start your WebLogic server and open the WebLogic system console. The system console can be accessed using a URL similar to the following:

```
http://<hostname or ip-address>:<port number of your  
WebLogic Server>/console
```
3. Log in to the console using the system administrator ID and password for your WebLogic server.
4. In the Domain Structure panel, click **Deployments**.
If there are existing deployments of the application, stop them and delete them:
 - a. Stop the existing deployments of the application:
 - Check the box of the applicable deployment you want to delete.
 - Click **Stop** and select **Force Stop Now** from the pop-up menu.
 - In Delete Application Assistant, click **Yes**.
 - In Messages, the message "Selected Deployments have been requested to stop" displays.
 - b. Delete the existing deployments of the application:
 - Check the box of the applicable deployment you want to delete.
 - Click **Delete**.
 - In Delete Application Assistant, click **Yes**.
 - In Messages, this message displays: "Selected Deployments were deleted. Remember to click Activate Changes after you are finished."
 - In the Change Center panel, click **Activate Changes**.
5. In Location, browse to the directory where the `<application_name>.war` file was extracted and click **Next**.
6. Select **Install this deployment as an application** and click **Next**.
7. In Source accessibility, Select **I will make the deployment accessible from the following location**.
In Location:, make sure that the location points to the directory where the `<application_name>.war` file was extracted.
8. Copy the `weblogic.xml` file from the `INSTALL_DIR/repository/eardata/platform/descriptors/weblogic/WAR/WEB-INF` directory to the `INSTALL_DIR/extensions/smcsfs` directory.
9. Copy the `ycpapibundle.properties` file and `ycpapibundle_<lang>_country.properties` (if applicable) from the `<INSTALL_DIR>/resources` directory to the `<INSTALL_DIR>/repository/eardata/smcsfs/war/yfscommon` directory.

10. Copy the yscapibundle.properties file and yscapibundle_<lang>_<country>.properties (if applicable) from the <INSTALL_DIR>/resources directory to the <INSTALL_DIR>/repository/eardata/smcfs/war/yfscommon directory.
11. Copy the extnbundle.properties file and extnbundle_<lang>_<country>.properties (if applicable) from the <INSTALL_DIR>/resources/extn directory to the <INSTALL_DIR>/repository/eardata/smcfs/war/yfscommon directory.
12. (Optional) If a PCA is installed, copy the following files to the <INSTALL_DIR>/repository/eardata/smcfs/war/yfscommon directory:
 - <INSTALL_DIR>/resources/com.yantra.yfc.rcp.common_bundle.properties
 - <INSTALL_DIR>/resources/com.yantra.yfc.rcp_bundle.properties
 - <INSTALL_DIR>/resources/PCA_Codebundle.properties. For example, for Sterling Call Center and Sterling Store application, copy the ycdbundle.properties file.
13. Copy the extensions you have made to the <INSTALL_DIR>/extensions/global/webpages directory.

Exception: To include a customized JSP in a specific package, place it in <INSTALL_DIR>/extensions/<package>/webpages. For example, use <INSTALL_DIR>/extensions /smcfs/webpages and <INSTALL_DIR>/extensions/sbc/webpages for smcfs or sbc wars, respectively.

To include a customized JSP in all packages, place it in <INSTALL_DIR>/extensions /global/webpages.

14. Rebuild the EAR file as you did during installation process.
15. Extract the following war files from the smcfs.ear file:
 - smcfs.war
 - sbc.war
 - sma.war

Then extract each of these war files into a directory of your choice.
16. Extract the remaining jar files from the smcfs.ear file and copy all the extracted jar files to WEB-INF/lib. Doing this will make these jar files accessible to WebLogic and you don't have to include these jar files in the WebLogic CLASSPATH.
17. Deploy each directory on WebLogic as a Web application.
18. Test your customizations using the following WebLogic Hot Deployment Test Mode standards:

If you modify...	In these files...	Then...
Startup parameters	properties	Restart WebLogic
UI extensibility	JSP, JavaScript, CSS, theme XML	Load dynamically
Localization literals	alertmessages and localization bundle files	Restart WebLogic
Database extensions	entity XMLs	Rebuild the entities.jar file and include the jar in the classpath directory, then restart WebSphere.

If you modify...	In these files...	Then...
APIs and other template files	template XMLs	Rebuild the resources.jar file and include the jar in the classpath directory, then restart WebLogic.

What to do next

Sterling Selling and Fulfillment Foundation does not support documentation extensions for Context-Sensitive Help in exploded mode. If you want to use Context-Sensitive Help in exploded mode, build a separate smcfsdocs.ear file and deploy it.

Now you need to configure WebLogic as described subsequently. If you need further information, see the WebLogic documentation.

WebLogic must be configured to enable the server to read from the directory where the *application_name.war* file was extracted. The necessary steps for configuring WebLogic to run the application in exploded (non-EAR) mode for your development environment are given as follows.

Note:

Sterling Selling and Fulfillment Foundation deployed in exploded mode works in the same way as the Solution deployed in EAR mode. There are no performance implications specific to exploded mode.

IBM recommends the EAR mode of deployment in production. In case an application server hosts multiple applications, there is no interference for jars or classes across applications. This is because each application is packaged or deployed as a single EAR file isolated from other application packages (EARs). However, in exploded mode, the class that is first added to the classpath is always considered.

Preparing the Development Environment on WebSphere

About this task

When using WebSphere, you can test the modifications that you have made to Sterling Selling and Fulfillment Foundation.

Note: IBM recommends that you directly copy all the jar files from the extracted EAR to the WEB-INF/lib directory. Doing this will make these jar files accessible to WebSphere and you do not have to include these jar files in the WebSphere CLASSPATH.

To configure WebSphere to run the application in exploded mode:

Procedure

1. Set the following JVM arguments for this deployment:
 - Dsci.opsproxy.disable=Y
 - Dvendor=shell -DvendorFile=/servers.properties
2. Deploy the EAR, using the documentation provided by IBM. During deployment, WebSphere copies all the contents of WAR and EAR files to the

<WAS_HOME>/AppServer/profiles/<PROFILE_NAME>/installedApps/
<CELL_NAME>/<APP_NAME>/ directory.

After deployment, any files copied to the <WAS_HOME>/AppServer/profiles/
<PROFILE_NAME>/installedApps/<CELL_NAME>/<APP_NAME>/ directory
can be modified as needed. For example, if you are extending a custom code
written as part of the database extensibility, the custom code files can be
directly moved to the appropriate directory under <WAS_HOME>/AppServer/
profiles/<PROFILE_NAME>/installedApps/<CELL_NAME>/<APP_NAME>/
directory for testing. IBM calls this ability to modify and move files as needed
"hot deployment."

The custom JSPs written as part of UI extensibility can be directly incorporated
into the application WAR file.

Note: The application does not support documentation extensions for
Context-Sensitive Help in exploded mode. If you want to use Context-Sensitive
Help in exploded mode, build a separate smcfsdocs.ear file and deploy it.

3. Build your extensions.
4. Stop the application server.
5. Copy the jars created as part of building and deploying extensions and
overwrite the jars in <WAS_HOME>/AppServer/profiles/<PROFILE_NAME>/
installedApps/<CELL_NAME>/<APP_NAME>.

For example:

- If you are extending your database, build and deploy the entities.jar and
copy the jar to the <WAS_HOME>/AppServer/profiles/<PROFILE_NAME>/
installedApps/CELL_NAME/<APP_NAME> directory.
 - If you are extending UI resources, build and deploy the resources.jar, and
copy the jar to the <WAS_HOME>/AppServer/profiles/<PROFILE_NAME>/
installedApps/<CELL_NAME>/<APP_NAME> directory.
6. Copy your customized files (for example, localization literal files, JSPs), to the
appropriate <WAS_HOME>/AppServer/profiles/<PROFILE_NAME>/
installedApps/<CELL_NAME>/APP_NAME/smcfs.war/<Module_Name>
directory.

For example, if you have some customizations in the Catalog module, add the
files in the <WAS_HOME>/AppServer/profiles/<PROFILE_NAME>/
installedApps/<CELL_NAME>/<APP_NAME>/smcfs.war/ycm directory.

7. Restart the application server.
8. Test your customizations using the following WebSphere Hot Deployment Test
Mode standards:

If you modify...	In these files...	Then...
Startup parameters	properties	Restart WebSphere.
UI extensibility	JSP, JavaScript, CSS, theme XML	Load dynamically.
Localization literals	alertmessages and localization bundle files	Restart WebSphere.

If you modify...	In these files...	Then...
Database extensions	entity XMLs	Rebuild the entities.jar file. Include the jar in the following directory: <i>WAS_HOME/AppServer/profiles/PROFILE_NAME/installedApps/CELL_NAME/APP_NAME</i> Restart WebSphere.
APIs and other template files	template XMLs	Rebuild the resources.jar file. Include the jar in the following directory: <i>WAS_HOME/AppServer/profiles/PROFILE_NAME/installedApps/CELL_NAME/APP_NAME</i> Restart WebSphere.

Preparing the Development Environment on JBoss

About this task

This section explains how to test the modifications made to Sterling Selling and Fulfillment Foundation when using JBoss.

Note: IBM recommends that you directly copy all the jar files from the extracted EAR to the WEB-INF/lib directory. Doing this will make these jar files accessible to JBoss and you do not have to include these jar files in the JBoss CLASSPATH.

To configure JBoss to run the application in exploded mode:

Procedure

1. Edit the `<JBOSS_DOMAIN>/bin/run.cmd` script for Windows (run.sh for UNIX), and set the following argument in Options as Java parameters:

```
-Dsci.opsproxy.disable=Y -Dvendor=shell
-DvendorFile=/servers.properties
```

2. Rebuild the EAR file as you did during the installation process.

Note: The application does not support documentation extensions for context-sensitive Help in exploded mode. If you want to use context-sensitive Help in exploded mode, build a separate smcfsdocs.ear file and deploy it.

3. Stop the application server and execute the following steps.
For example, if you have some customizations in the Catalog module, add the files in the `<JBOSS_HOME>/server/<SERVER_NAME>/deploy/smcfs.ear/smcfs.war/ycm` directory.
Create a new directory and name it smcfs.ear.
4. Extract the EAR into the smcfs.ear directory you created.
5. Within the smcfs.ear directory, a smcfs.war file exists. Rename this .war file or copy it into another directory.
6. Within the smcfs.ear directory, create a new subdirectory, and name it smcfs.war.
7. Extract and extract all the files from the smcfs.war file into the smcfs.ear/smcfs.war subdirectory.
8. Delete the smcfs.war file that you renamed or copied in step 5.

9. Copy the jars created as part of building and deploying extensions and overwrite the jars in `<JBoss_HOME>/server/<SERVER_NAME>/deploy/smcfs.ear` directory.
For example:
 - If you are extending your database, build and deploy the `entities.jar` and copy the jar to the `<JBoss_HOME>/server/<SERVER_NAME>/deploy/smcfs.ear` directory.
 - If you are extending UI resources, build and deploy the `resources.jar`, and copy the jar to the `<JBoss_HOME>/server/<SERVER_NAME>/deploy/smcfs.ear` directory.
10. Copy your customized files (for example, localization literal files, JSPs), to the appropriate `<JBoss_HOME>/server/<SERVER_NAME>/deploy/smcfs.ear/smcfs.war` directory.
11. Restart the application server.
12. After deploying, you can modify the files copied to the `<JBoss_HOME>/server/<SERVER_NAME>/deploy` directory. For example, if you extend a custom code written as part of database extensibility, you can directly move the extended custom code to the appropriate directory under the `<JBoss_HOME>/server/<SERVER_NAME>/deploy` directory for testing. JBoss identifies the changes and redeploys the application (Hot Deployment).
13. Test your customizations using the JBoss Hot Deployment Test Mode standards described in the following table.

If you modify...	In these files...	Then...
Startup parameters	properties	Restart JBoss.
UI extensibility	JSP, JavaScript, CSS, theme XML	Load dynamically.
Localization literals	alertmessages and localization bundle files	Restart JBoss.
Database extensions	entity XMLs	Rebuild the <code>entities.jar</code> file and include the jar in the <code>JBoss_HOME/server/SERVER_NAME/deploy/smcfs.ear</code> directory, then restart JBoss.
APIs and other template files	template XMLs	Rebuild the <code>resources.jar</code> file and include the jar in the <code>JBoss_HOME/server/SERVER_NAME/deploy/smcfs.ear</code> directory, then restart JBoss.
Application configuration files	web.xml, application.xml, and property files	Restart JBoss.

Developing and Testing in the Development Environment

Now that you have set up your development environment, you are ready to begin customizing Sterling Selling and Fulfillment Foundation, using the directions provided throughout this guide. This section explains how to use the icons that enable you immediately test UI customization.

Note: After any modifications are made to the following files, the application server must be restarted:

- `datatypes.xml` file

- yfsdatatypemap.xml file



Testing UI Customizations

About this task

After making changes to UI Resources within the Resource Hierarchy tree, you can test your changes immediately by using the Cache Refresh icons.

To use the cache refresh icons:

Procedure

1. Make changes as needed.
2. Select the refresh cache icon that fits your needs as follows:
 - If you want to update one entity and its child resources - Select the specific entity and select the  Refresh Entity Cache icon
 - If you want to update all resources - Select the  Refresh Cache icon
3. Log into the application again to test your changes.

Configuring the UI Cache Refresh Actions

In a standard deployment of Sterling Selling and Fulfillment Foundation, any configuration changes made within the Resource Hierarchy tree do not take effect until the application server has been restarted. This means that testing your UI extensions can be a time-consuming activity. Therefore, the application provides actions within the Resource Hierarchy tree that enable you to refresh the resources so that your modifications can be tested immediately.



These actions can only be enabled in a development environment. They do not work in a deployment environment. This section explains how to enable these actions. These actions should be disabled in a deployment environment.

Configure Resource Cache Refresh Actions

About this task

To configure the resource cache refresh actions:

Procedure

1. Use the `<INSTALL_DIR>/properties/customer_overrides.properties` file to set the `yfs.uidev.refreshResources` property to `Y`.
2. This enables the following actions on the Resource Hierarchy tree in the Applications Manager:
 -  Refresh Cache icon - Refreshes all resources.
 -  Refresh Entity icon - Refreshes the selected entity resource and its child resources.

For instructions on using the Refresh Cache icons, see Testing UI Customizations.

Chapter 4. Customization Using Microsoft COM+

Microsoft COM+ Prerequisites

When using Microsoft COM+, you need to create and configure the application on a Windows server. You also need to create and install a client proxy.

Creating a COM+ Application on Windows

About this task

To create the Sterling Selling and Fulfillment Foundation COM+ application on a server that has a Windows operating system:

Procedure

1. From the Windows Start menu, navigate to **Administrative Tools** → **Component Services**.
2. From the Component Services tree, navigate to **Component Services** → **Computers** → **My Computer** → **COM+ Applications** and then right-click **COM+ Application**. Select **New** → **Application**.
3. After the Welcome to COM Application Install Wizard screen appears, click **Next**.
4. Select **Create an Empty Application**.
5. In the Create Empty window, enter the following and then click **Next**:
 - For Application Name, enter your application name.
 - For Activation Type, select **Server Application**. This ensures that the components are started as dedicated processes.
6. In the Set Application Identity window, select **This User** and enter the appropriate Windows user name and password. This user is the identity under which the application is run. Make sure that the user belongs to the Administrators group. Click **Next** and then click **Finish**.

The newly created COM+ application appears under the Component Services tree.

Results

Now you can add components to your w COM+ application.

Adding Components to a COM+ Application

About this task

To add components to a Sterling Selling and Fulfillment Foundation COM+ application:

Procedure

1. From the Component Services tree, navigate to **Component Services** → **Computers** → **My Computer** → **COM+ Applications** → **Selling and Fulfillment Foundation** → **Components** and then right-click **Components**. Select **New** → **Components**.

2. After the Welcome to COM Component Install Wizard screen appears, click **Next**.
3. Select **Install New Component**.
4. Browse to *INSTALL_DIR*/bin/YIFComApi.dll. Select it and select **Open**. Then click **Next** and **Finish**.
5. Make sure that your system path contains the directories that store the following DLLs:
 - <INSTALL_DIR>/bin/YIFJNIApi.dll
 - <INSTALL_DIR>/bin/release/msvcrt.dll
 - <INSTALL_DIR>/bin/release/msvcp60.dll
 - jvm.dll (usually found under <JAVA_HOME>/jre/bin/hotspot on the client machine)
6. Verify that the <INSTALL_DIR>/properties directory includes the yfs.properties file and the <INSTALL_DIR>/resources directory includes the yifclient.properties file.

Note: If you have added any entries to the *INSTALL_DIR*/properties/customer_overrides.properties file, ensure that this file is included in the <INSTALL_DIR>/properties directory.

Configuring the COM+ Service

About this task

To configure the Sterling Selling and Fulfillment Foundation COM+ service:

Procedure

1. From the Component Services tree, right-click the newly created COM+ Application.
2. Select **Properties**.
The Properties dialog box is displayed.
3. Select the **Advanced** tab.
4. Under Server Process Shutdown panel, select **Minutes Until Idle Shutdown**, enter the time in minutes after which you want the process to shut down, and then click **OK**.
5. Double-click the **COM+** application.
6. Double-click **Components**.
7. Right-click **YIFComApi.YIFComApi.1** and select **Properties**.
The YIFComApi.YIFComApi.1 Properties dialog box appears.
8. Select the **Activation** tab.
9. Select **Enable object pooling**.
10. In the Object pooling section, enter the **Minimum and Maximum pool** sizes based on the Component usage. Configure pooling to make optimal use of your hardware resources. The pool configuration can change as available hardware resources change.
11. Select **Enable Just In Time Activation**.
JIT activation activates an instance of an object just before the first call is made to it and then immediately deactivates the instance after it finishes processing its work.

Creating a Client Proxy

About this task

To create a client proxy:

Procedure

1. Right-click your Sterling Selling and Fulfillment Foundation COM+ application and select **Export**.
2. After the Welcome to COM Application Export Wizard screen appears, click **Next**.
3. In the Application Export window, enter the path and file name where the export .MSI file is to be created.
4. Select **Export as Application Proxy**.
5. Click **Next** and then click **Finish**.

Installing a Client Proxy

About this task

To install a client proxy, make sure the following DLL files are in your system path:

- <INSTALL_DIR>/bin/YIFJNIApi.dll
- <INSTALL_DIR>/bin/release/msvcrt.dll
- <INSTALL_DIR>/bin/release/msvcp60.dll
- jvm.dll (located under the *JAVA/jre/bin/hotspot* directory on the client machine. Double-click the *.MSI file to install the component on the client.)

Note: This method of including the DLL files is deprecated in Release 7.7. The recommended method to call APIs and Services from a Microsoft Windows COM+ environment is through Web services or over HTTP.

Chapter 5. Masking Sensitive Information During Logging

Masking Sensitive Information During Logging Using Log4j

About this task

You can configure the log4j utility to prevent sensitive information such as credit card number, passwords, and so forth from being logged in the log messages. To mask the sensitive information, you must use the application-provided custom log4j Layout and Filter and also define a set of named regular expressions in the `customer_override.properties` file.

The custom log4j layout will get the formatted message and filter the results based on a set of configurable regular expressions. This custom log4j filter will allow you to match the message against a set of regular expressions and discard the message, if it matches.

To mask sensitive information during logging:

Procedure

1. Change the layout class name in the custom logging configuration to `SCIFilteredPatternLayout`. For example:

```
<layout
class="com.sterlingcommerce.woodstock.util.frame.logex.SCIFilteredPatternLayout"
  >
  <param name="ConversionPattern" value="%d:%-7p:%t: %-60m
[%X{AppUserId}]: %-25c{1}%n"/>
  <param name="FilterSet" value="common-filter"/> <!-- Optional -->
</layout>
```

2. Change the filter class name in the custom logging configuration to `SCIPatternFilter`. For example:

```
<filter
class="com.sterlingcommerce.woodstock.util.frame.logex.SCIPatternFilter">
  <param name="FilterSet" value="suppress" /> <!-- Optional -->
</filter>
```

3. Define a set of named regular expressions against which you want to match the message in the `<INSTALL_DIR>/properties/customer_ overrides.properties` file using following properties:

```
filterset.<name>.pattern.<num>=<pattern>
```

This property is optional:

```
filterset.<name>.replace.<num>=<replace>
```

where `<pattern>` is a Java-style regular expression and defines the regular expression against which you want to match the message string. The `replace` property is optional, and defines the string which will be used to replace the expression.

You can set the default `FilterSet` parameters by setting the following properties:

```
default.filter.filterset=<filter_name>
```

```
default.layout.filterset=<layout_name>
```

You can also define a common set of regular expression patterns across multiple filter sets as following:

```
filterset.name.includes=<name1>,<name2>,...
```

You can view the `<INSTALL_DIR>/properties/logfilter.properties.in` file to see some sample entries for defining these properties.

Chapter 6. Data Validation

About Data Validation

The Sterling Selling and Fulfillment Foundation provides the Data Validation functionality for validating and sanitizing request inputs and outputs. You can use the Data Validation functionality to allow only explicitly defined characteristics in the input and output requests, and drop all the other data. You can define your own validation rules for validating different request parameters. You can also encode data before sending it back to the user interface (UI).

Data validation or sanitization can be performed for various kinds of inputs such as parameter name, parameter value, cookie name, cookie value, and so on. The application also supports regular expression based validation.

Input Validator

The Input Validator finds all the validation rules that are registered for a particular input, and performs the validation. The Validator is called by a request wrapper to validate request inputs.

By default, to validate request inputs such as parameter value, parameter name, and so on, the Input Validator uses the regular expressions shipped out-of-the-box by the application. The out-of-the-box shipped regular expressions are defined in the `regularexpressions/sc_regularexpressions.xml` file ((located inside the `<INSTALL_DIR>/jar/platform_afc/5_7/platform_dv.jar`).

Note: If you want to relax some of the validation rules for certain inputs, you must register all the custom validation rules with the Input Validator.

Validation Rule

A validation rule performs validation and sanitization of the input. A validation rule contains a property as input identifier for which validation has to happen. A validation is invoked whenever the corresponding input request is accessed. A validation rule must specify the name of the input, it has to validate. For example, to validate the value of a parameter, the validation rule must specify the name of that particular parameter. Multiple inputs with the same name can exist. All the validation rules must be registered with the Input Validator in order to validate the corresponding input.

Some validation rules are shipped out-of-the-box by the application. The out-of-the-box shipped validation rules are defined in the `validationrules/sc_validationrules.xml` file (located inside the `<INSTALL_DIR>/jar/platform_afc/5_7/platform_dv.jar`). These validation rules are invoked for all the inputs belonging to the same category. For example, all the HTTP Header names are validated against the `HTTPHeaderName` regular expression.

Note: No validation rules are defined for a given input. The validation rules specified earlier will be used to validate the input.

You can define the following types of validation rules:

- Regular Expression-Based Validation Rule—This type of validation rule is designed to perform regular expression-based validations. This validation rule type supports multiple whitelist and blacklist regular expressions.
- Java-Based Validation Rule—This type of validation rule is designed to perform Java-based validation and sanitization of inputs. This validation rule type validates an input and then calls the `getValidInput()` method of the implementation class.

Disabling Data Validation

About this task

By default, data validation is enabled on all input requests, which are validated against the registered validation rules. You can disable the data validation on input requests by adding a context parameter in the module configuration file.

To disable data validation, in the `web.xml` file located in your `EARFILE/WARFILE/WEB-INF` folder, add an entry for the `context-param` element as follows:

```
<context-param>
  <param-name>scui-suppress-request-validation</param-name>
  <param-value>TRUE</param-value>
</context-param>
```

Bypassing Data Validation for an URI

By default, data validation is enabled on all input requests. You can, however, bypass data validation on input requests for some specific Universal Resource Indicators (URIs) by adding a bypass URI as context parameters in the module configuration file.

To bypass data validation, in the `web.xml` file located in your `EARFILE/WARFILE/WEB-INF` folder, add an entry for the `config-param` element for each such URI, for example:

```
<context-param>
  <param-name>request.validation.bypass.uri.1</param-name>
  <param-value>/console/login.jsp</param-value>
</context-param>
<context-param>
  <param-name>request.validation.bypass.uri.2</param-name>
  <param-value>/console/start.jsp</param-value>
</context-param>
<context-param>
  <param-name>request.validation.bypass.uri.endswith.1</param-name>
  <param-value>.js</param-value>
</context-param>
<context-param>
  <param-name> request.validation.bypass.uri.regex.1</param-name>
  <param-value>^.test.jsp$</param-value>
</context-param>
```

These context parameters can have names starting with `request.validation.bypass.uri`, or `request.validation.bypass.uri.endswith`, or `request.validation.bypass.uri.regex`, as described in the following list. You can define multiple entries for these context parameters.

- `request.validation.bypass.uri`—Any request with an URI that is the same as the value specified in the `param-value` element of the context parameter will be bypassed and not validated.

- `request.validation.bypass.uri.endswith`—Any request with an URI that ends with the value specified in the `param-value` element of the context parameter will be bypassed and not validated.
- `request.validation.bypass.uri.regex`—Any URI request that matches the regular expression, as specified in the `param-value` element of the context parameter, will be bypassed and not validated.

Implementing Data Validation

To implement data validation, perform the following tasks:

- Defining Regular Expressions in Datatypes XML File
- Defining Regular Expressions in XML Files
- Registering Regular Expressions
- Defining Validation Rules in Datatypes XML File
- Externalizing Validation Rules Defined in the Datatypes XML File
- Defining Validation Rules in XML Files
- Defining Abstract Validation Rules
- Extending Abstract Validation Rules
- Registering Validation Rules
- Overriding Regular Expressions
- Overriding Validation Rules
- Defining Error Messages
- Localizing Error Messages
- Defining Custom Regular Expression Error Message Provider
- Localizing Validation Rules

Defining Regular Expressions in Datatypes XML File

You can define the regular expressions for input validation in the `datatypes.xml` file. It is recommended that you define those regular expressions in the `datatypes.xml` file, which are being used for a single datatype and are not being used for multiple datatypes at the same time. Otherwise, if you define regular expression for individual data types you want to use the same regular expression in multiple data types, then making a change to such regular expression may become cumbersome because you will have to make the similar change at multiple places manually. You can define regular expression for input validation in the `datatypes.xml` file in the following format:

```
<DataType Name="Address" Size="70" Type="NVARCHAR">
  <UIType Size="30" UITableSize="30"/>
  <Validation>
    <Regex MaxLength="200" JavaPattern="^[a-zA-Z4-9.@\-\/+=_()
      \{\}\[\],:&apos;&quot;]*$" JSPattern="" />
  </Validation>
</DataType>
```

Note: You can externalize the regular expressions defined in the `datatypes.xml` file into a separate XML file and then use reference of these regular expressions in multiple datatypes.

Defining Regular Expressions in XML Files

You can define regular expressions for input validation in separate XML files and reference to these regular expressions can be used in multiple datatypes defined in the `datatypes.xml` file and also in multiple rules defined in the different rules XML

files. The main advantage of this approach is that it makes it easier to modify the regular expression in one place and the changes will take effect both in the datatypes.xml file and different rules XML files. Each regular expression has an unique id associated with it. The regular expression files can be registered with the Sterling Selling and Fulfillment Foundation by adding a context parameter in the web.xml file.

To define regular expression for input validation, create an XML file in the following format:

```
<RegularExpressions>
  <RegularExpression id="" javaPattern="" jsPattern="" blacklistErrorMsg=""
    whitelistErrorMsg=""/>
  <RegularExpression id="" javaPattern="" jsPattern=""/>
</RegularExpressions>
```

The following table describes various elements and attributes of the regular expressions XML file.

Note: Any empty attribute will be considered as not provided. For example, an element defined as `<RegularExpression id="dates" javaPattern="" jsPattern="^[a-zA-Z0-9.,!\-/+=_:]*$"/>` is equivalent to `<RegularExpression id="dates" jsPattern="^[a-zA-Z0-9.,!\-/+=_:]*$"/>`.

Element/Attribute	Description
RegularExpressions	Required. The regular expressions XML file must have RegularExpressions as root element.
RegularExpression	Optional. Each regular expression must be defined as one RegularExpression element. It can have zero or more occurrences.
id	Required. Unique identifier for the regular expression. This id can be used as reference for this regular expression in various places.
javaPattern	Optional. Regular Expression pattern for performing server-side validation. If not specified, the input validation happens on the client side based on the jsPattern attribute value and the server-side validation is not done.
jsPattern	Optional. Regular Expression pattern for performing client-side validation. If not specified, the input validation happens on the server side based on the javaPattern attribute value and the client-side validation is not done.
whitelistErrorMsg	Optional. Bundle key for the error message to be displayed, if the white list patterns of the regular expression fail.
blacklistErrorMsg	Optional. Bundle key for the error message to be shown if the black list patterns of the regular expression fail.

For example, let us consider that you define a regular expression with an id dates in the regular expressions XML file as follows:

```
<RegularExpressions>
  <RegularExpression id="dates" javaPattern="^[a-zA-Z0-9.,!\-/+=_:]*$"
    jsPattern="^[a-zA-Z0-9.,!\-/+=_:]*$" blacklistErrorMsg=""
    whitelistErrorMsg=""/>
</RegularExpressions>
```

Now, you can reference this dates regular expression in both Rule XML as well as in the Datatypes XML.

Using Regular Expression Reference in Rule XML

You can provide a reference of the regular expression id in the RegularExpression element when defining a rule in the rule XML. For example,

```
<ValidationRules>
  <Rule id="" ruleType="Regex" inputType="" inputName="" uri=""
    maxLength="" minLength="" allowNull="" >
    <Whitelist>
      <RegularExpression ref="dates"/>
    </Rule>
</ValidationRules>
```

Using Regular Expression Reference in Datatypes XML

You can provide a reference of the regular expression id in the Regex element when defining a regular expression in the datatypes.xml file. For example,

```
<DataTypes>
  <DataType Name="Date" PpcSize="12" Size="7" Type="DATE">
    <Validation>
      <Regex Ref="dates" />
    </Validation>
    <UIType Size="8" UITableSize="15"/>
  </DataType>
</DataTypes>
```

Registering Regular Expressions

To register regular expression files with Sterling Selling and Fulfillment Foundation, add a new context parameter for each regular expressions file in the web.xml file (located inside your EARFILE/WARFILE/WEB-INF directory) in one of the following ways:

```
<context-param>
  <param-name>scui-regex-file-unique-identifier</param-name>
  <param-value><file-path-inside-webapp>::<load order></param-value>
</context-param>
```

OR

```
<context-param>
  <param-name>scui-regex-file-unique-identifier</param-name>
  <param-value><fully-qualified-name-of-the-file>::<load order></param-value>
</context-param>
```

where *<load order>* defines the order in which the regular expressions file must be loaded. File having the least load order will be read first. Multiple files can have same load order. If no load order is defined, the system considers it to be zero.

Note: You must use a load order above 500 to register the regular expression files.

For example,

```
<context-param>
  <param-name>scui-regex-file-fwk-1</param-name>
  <param-value>/WEB-INF/regex1.xml</param-value>
</context-param>
<context-param>
  <param-name>scui-regex-file-fwk-2</param-name>
  <param-value>/WEB-INF/regex2.xml::1</param-value>
</context-param>
<context-param>
  <param-name>scui-regex-file-fwk-3</param-name>
  <param-value>/com/test/regex-used-from-datatypes.xml::2</param-value>
</context-param>
```

Note: The regular expression files containing regular expressions, which are being referenced from the datatypes.xml file, must be kept inside a JAR and the JAR must be available in APP dynamic classpath. The regular expression files containing other rules (which are not being referenced from datatypes.xml file) must be kept inside the <INSTALL_DIR>/repository directory. IBM recommends that you keep such files inside the EARFILE/WARFILE/WEB-INF directory to make them inaccessible by http access.

Defining Validation Rules in Datatypes XML File

You can register a validation rule using the datatypes.xml file. This method of registering a validation rule can only be used for parameter value inputs. The datatype for a parameter is deduced using the datatypes map. And the parameter value is validated using the validation rules registered against that datatype.

Applications based on the HTML UI Framework can register a regular expression based or java based validation rule in the datatypes.xml file in the following way:

```
<DataType Name="Address" Size="70" Type="NVARCHAR">
  <UIType Size="30" UITableSize="30"/>
  <Validation>
    <Regex JavaPattern="<pattern>" JSPattern="<pattern>" allowNull="false"/>
    <Impl JavaClass="com.sterlingcommerce.test.MyRuleClass"
      JSFunctionName="myJavascriptFunction"/>
  </Validation>
</DataType>
```

By default, for a <Regex> element, the maximum size of the validation rule is set to the size of the datatype. You can override the maximum size of the validation rule using the MaxLength attribute. Also, you can set the minimum size of the validation rule using the MinLength attribute. Similarly, you can override other attributes such as JSPattern, JavaPattern, and AllowNull. For example,

```
<DataType Size="5" Name="Pincode" Type="NUMBER">
  <Validation>
    <Regex MaxLength="200" MinLength="3" JavaPattern="^[a-zA-Z0-9.,!\-/+=:]*$"
      JSPattern="^[a-zA-Z0-9.,!\-/+=:]*"
  </Validation>
</DataType>
```

Note: Java based validation rule class must have a fully qualified class name. And this class must implement the ISCValidationRule interface.

Note: In the datatypes.xml file, you can also define javascript patterns and functions to validate the input on the client itself. These client side validations will be fetched on the client and all the corresponding inputs will be validated against these client side validations.

Externalizing Validation Rules Defined in the Datatypes XML File

You can also externalize the validation rules defined in the datatypes.xml file as abstract rules. Firstly you need to define such rules as abstract rules and then extend this abstract rule in the datatypes.xml file.

For example, let us consider that you have defined a validation rule in the datatypes.xml file as follows:

```
<DataTypes>
  <DataType Name="Date" PpcSize="12" Size="7" Type="DATE">
    <Validation>
      <Regex MaxLength="200" JavaPattern="^[a-zA-Z0-9.,!\-/+=:]*"
        JSPattern="^[a-zA-Z0-9.,!\-/+=_]*"
    </Validation>
  </DataType>
</DataTypes>
```

```

        </Validation>
        <UIType Size="8" UITableSize="15"/>
    </DataType>
</DataTypes>

```

You can externalize this validation rule as an abstract rule in the Rule XML file as follows:

```

<ValidationRules>
    <Rule id="abstract1" ruleType="Regex" abstract="true" maxLength="100" minLength="0">
        <Whitelist>
            <RegularExpression ref="ref1"/>
        </Whitelist>
    </Rule>
</ValidationRules>

```

Now, you can reference this abstract validation rule (abstract1) in the datatypes.xml file by adding the extends attribute in the Rule element. For example,

```

<DataTypes>
    <DataType Name="Date" PpcSize="12" Size="7" Type="DATE">
        <Validation>
            <Rule Extends="abstract1" />
        </Validation>
        <UIType Size="8" UITableSize="15"/>
    </DataType>
</DataTypes>

```

Defining Validation Rules in XML Files

Validation rules that are not related to datatypes or parameter value validation cannot be defined in the datatypes.xml file. But you can define such rules by creating a new Rule XML files. The rules XML files can be registered with the Sterling Selling and Fulfillment Foundation by adding a context parameter for each rule XML file in the web.xml file.

To define a rule for input validation, create an XML file in the following format:

```

<ValidationRules>
    <Rule id="" ruleType="Regex" inputType="" inputName="" uri="" maxLength=""
        minLength="" allowNull="" >
        <Whitelist>
            <RegularExpression ref="" />
        </Whitelist>
        <Blacklist>
            <RegularExpression ref="" />
        </Blacklist>
    </Rule>
    <Rule id="" ruleType="Java" inputType="" inputName="" uri="" impl="" />
    <Rule id="abstract1" ruleType="Java" abstract="true" impl="" />
    <Rule id="" extends="abstract1" inputType="" inputName="" uri="" />
    <Rule id="abstract2" ruleType="Regex" abstract="true" maxLength=""
        minLength="" allowNull="" >
        <Whitelist>
            <RegularExpression ref="" />
        </Whitelist>
        <Blacklist>
            <RegularExpression ref="" />
        </Blacklist>
    </Rule>
    <Rule id="" extends="abstract2" inputType="" inputName="" uri="" maxLength=""
        minLength="" allowNull="" />
</ValidationRules>

```

The following table describes various elements and attributes of the rules XML file.

Note: Any empty attribute will be considered as not provided. For example, an element defined as `<Rule id="dates" ruleType="Regex" inputType="" inputName="" uri="" maxLength="" minLength="" allowNull="" >` is equivalent to `<Rule id="dates" ruleType="Regex" inputName="" >`. An exception to this behavior is attribute `inputName`. The `inputName` attribute can contain empty string and it is a valid value for `inputName` attribute.

Element/Attribute	Description
ValidationRules	Required. The rules XML file must have ValidationRules as root element.
Rule	Optional. Each rule (either java rule or regular expression rule) must be defined as one Rule element. It can have zero or more occurrences.
id	Optional. Unique identifier for the rule. Note: This attribute is required, if you are extending an existing rule. If not provided, the existing rule cannot be extended. If a duplicate id found, and it does not result into extension, an exception is thrown.
abstract	Optional. Set the value of this attribute to true, if the rule is an abstract rule. An abstract rule can be extended by another rule.
extends	Optional. Identifier of the abstract rule being extended.
ruleType	Required. Type of rule. Valid Values: Java, Regex. Note: If the rule is extending an abstract rule, this attribute should not be present.
inputType	Required. Type of the input. Valid Values: HTTPParameterValue, HTTPParameterName, HTTPCookieValue, HTTPCookieName, HTTPHeaderValue, HTTPHeaderName, HTTPScheme, HTTPServerName, HTTPContextPath, HTTPPath, HTTPQueryString, HTTPURI, HTTPURL, HTTPJSESSIONID, HTTPServletPath, JavascriptClient. Note: The JavascriptClient input type is used to fetch validation rules on the client for inputs, which do not have any datatype associated with them. Note: For input type JavascriptClient, only global or default rules can be defined. Therefore, valid values for attribute <code>inputName</code> are <code>_global_</code> or <code>_default_</code> . Value of attribute <code>uri</code> is ignored. The <code>impl</code> attribute should contain the name of the javascript method. The regular expression defined by attribute <code>ref</code> should have a javascript regular expression.
inputName	Note: For abstract rules, this attribute should not be present. Required. Name of the input.
uri	Note: For abstract rules, this attribute should not be present. Optional. URI for which the rule should be valid.
Child Elements and Attributes for rule type Java	Note: For abstract rules, this attribute should not be present.
impl	Required. Fully qualified name of the rule implementation class. This class must extend the <code>ISCVailidaitonRule</code> class. Note: If the rule is extending an abstract rule, this attribute should not be present.

Element/Attribute	Description
Child Elements and Attributes for rule type Regex	
maxLength	Optional. Allowed maximum length. Valid Values: number, string. Note: If the rule is extending an abstract rule, this value will override the maxLength defined for an abstract rule.
minLength	Optional. Allowed minimum length. Valid Values: number, string. Note: If the rule is extending an abstract rule, this value will override the minLength defined for an abstract rule.
allowNull	Optional. Set the value of this attribute to false, if you want to mandate the value for the input. Valid Values: true, false. Note: If the rule is extending an abstract rule, this value will override the allowNull defined for an abstract rule.
Whitelist	Optional. Container element for white list patterns. It can have zero or one occurrence. Note: If the rule is extending an abstract rule, this attribute should not be present.
Blacklist	Optional. Container element for black list patterns. It can have zero or one occurrence. Note: If the rule is extending an abstract rule, this attribute should not be present.
RegularExpression	Optional. This element can have zero or many occurrences.
ref	Required. Reference of the regular expression definition.

Defining Abstract Validation Rules

An abstract validation rule does not have `inputType`, `inputName` and `uri` attributes. These attributes are provided by the validation rule extending the abstract validation rule. You can define a validation rule as an abstract validation rule by setting the value of abstract attribute to true. For example,

```
<ValidationRules>
  <Rule id="abstract1" ruleType="Regex" abstract="true" maxLength="10"
    minLength="0" allowNull="false" >
    <Whitelist>
      <RegularExpression ref="regex1" />
    </Whitelist>
  </Rule>
  <Rule id="abstract2" ruleType="Java" impl="com.sterling.validation.testRule"
    abstract="true">
  </Rule>
</ValidationRules>
```

Note: For JSON parameter values used in XAPI inputs in the Web UI Framework based applications, framework provides an abstract rule definition with id as `uifwkimpl-json-xapi-input-param-value`.

For all the inputs which are in JSON format and are being used in XAPI calls in the Web UI Framework based applications, you must extend the rule definition `uifwkimpl-json-xapi-input-param-value`. For example:

```
<Rule id="sampleRule1" extends="uifwkimpl-json-xapi-input-param-value"
inputType="HTTPParameterValue" inputName="getCategoriesList"/>
```

The abstract rule `uifwkimpl-json-xapi-input-param-value` is of type Java.

Extending Abstract Validation Rules

You can extend an abstract rule by adding the `extends` attribute in the `Rule` element. The value of the `extends` attribute should contain the identifier of the abstract rule that you want to extend. For example,

```
<ValidationRules>
  <Rule id="impl1" extends="abstract1" inputType="HTTPParameterValue"
        inputName="Test" uri="/console/home.do" maxLength="100">
  </Rule>
  <Rule id="impl2" extends="abstract2" inputType="HTTPParameterValue"
        inputName="Test1" uri="/console/home.do">
  </Rule>
</ValidationRules>
```

In this case, the `maxLength` defined for rule `impl1` will override the `maxLength` defined for rule `abstract1`.

Registering Validation Rules

To register the rule XML files with the Sterling Selling and Fulfillment Foundation, add a new context parameter for each rule XML file in the `web.xml` file (located inside your `EARFILE/WARFILE/WEB-INF` directory) in one of the following ways:

```
<context-param>
  <param-name>scui-validation-rules-file-unique-identifier</param-name>
  <param-value><file-path-inside-webapp>::<load order></param-value>
</context-param>
```

OR

```
<context-param>
  <param-name>scui-validation-rules-file-unique-identifier</param-name>
  <param-value>fully-qualified-name-of-the-file::load order</param-value>
</context-param>
```

where *load order* defines the order in which the rules file must be loaded. File having the least load order will be read first. Multiple files can have same load order. If no load order is defined, system considers it as zero.

Note: You must use load order of above 500 to register the validation rule files.

For example,

```
<context-param>
  <param-name>scui-validation-rules-file-fwk-1</param-name>
  <param-value>/WEB-INF/rules1.xml</param-value>
</context-param>
<context-param>
  <param-name>scui-validation-rules-file-fwk-2</param-name>
  <param-value>/WEB-INF/rules2.xml::1</param-value>
</context-param>
<context-param>
  <param-name>scui-validation-rules-file-fwk-3</param-name>
  <param-value>/com/test/rules-being-used-from-datatypes.xml::2</param-value>
</context-param>
```

Note: The validation rule files containing regular expressions, which are being referenced from the `datatypes.xml` file, must be kept inside a JAR and the JAR must be available in APP dynamic classpath. The regular expression files containing other rules (which are not being referenced from `datatypes.xml` file)

must be kept inside the `<INSTALL_DIR>/repository` directory. IBM recommends you to keep such files inside the `EARFILE/WARFILE/WEB-INF` directory to make them inaccessible by http access.

Overriding Regular Expressions

You can override an existing regular expression by defining and registering a similar entry in their custom regular expressions file. When defining the regular expression you must give the same id for the regular expression. You must register the new regular expression XML files with higher load orders.

Overriding Validation Rules

You can override an existing validation rule by defining and registering a similar entry in their custom rules file. When defining the rule you must give the same value for the following attributes:

- id
- inputName
- inputType
- url (if provided)

Note: The id attribute is the primary key for performing the similarity check on a validation rule. If the id does not match, or is not defined, the validation rule will not be extended. And if the id matches but other attributes for similarity doesn't match, an exception will be thrown.

Note: You can only override rules that are registered using rule XMLs and NOT `datatypes.xml` file. Also, for abstract rules only the id attribute is checked.

You must register the new rule XML files with higher load orders.

Note: When you are registering validation rules for inputs, the following `inputTypes` constants must be used:

- `HTTPParameterValue`
- `HTTPParameterName`
- `HTTPCookieValue`
- `HTTPCookieName`
- `HTTPHeaderValue`
- `HTTPHeaderName`
- `HTTPScheme`
- `HTTPServerName`
- `HTTPContextPath`
- `HTTPPath`
- `HTTPQueryString`
- `HTTPURI`
- `HTTPURL`
- `HTTPJSESSIONID`
- `HTTPServletPath`
- `JavascriptClient`

Defining an Adapter to Find Validation Rules

This is an optional task. You can define an adapter, to find the validation rules, that will be used to validate the parameter values. This adapter class must implement the `ISCUInputValidationAdapter` interface, and must be registered with the application as a context parameter. For example:

```
<context-param>
  <param-name>scui-param-value-validation-adapter</param-name>
  <param-value>test.MyParamValueValidationAdapter</param-value>
</context-param>
```

You must implement the `getValidationRules()` method of the `ISCUInputValidationAdapter` interface and pass the parameter name in the `name` argument.

When validating a parameter value, the system will call the registered adapter to find the rules against which the parameter value should be validated. The `getValidationRules()` method can either return all the rules registered for the passed parameter name, or have some logic to find other rules too. If no adapter is registered, the system will use all the rules registered for the given parameter name, along with the global rules or the default rules, to validate the parameter value.

Defining URI-Based Adapter to Find Validation Rules

This is an optional task. You can define the URI based adapter, which you want to use to find the validation rules, which will be used to validate the parameter values. This adapter class must implement the `ISCUInputValidationAdapter` interface and must be registered with the application as a context parameter as following:

```
<context-param>
<param-name>scui-param-value-validation-adapter::<uri-without-context-path></param-name>
<param-value><fully-qualified-adapter-class-name></param-value>
</context-param>
```

For example:

```
<context-param>
<param-name>scui-param-value-validation-adapter::/console/exception.list</param-name>
<param-value>test.Adapter</param-value>
</context-param>
```

You must implement the `getValidationRules()` method of the `ISCUInputValidationAdapter` interface and pass the parameter name in the `name` argument.

When validating a parameter value, the system will call the registered adapter to find out the URI based rules against which the parameter value should be validated. The `getValidationRules()` method can either return all the URI based rules registered for the passed parameter name, or can have some logic to find other rules also. If no adapter is registered, the system will use all the URI based rules registered for the given parameter name (along with global rules or default rules) to validate the parameter value.

Deleting Registered Validation Rules

You can delete the registered validation rules by calling any of the following methods of the `SCValidator` class:

- `removeDefaultRules (String inputType)`

- `removeGlobalRules` (String inputType)
- `removeRules` (String name, String inputType)
- `removeRules` (ISCValidationRuleKey name, String inputType)

Exception Handling

While validating a request, if an invalid input is found, an `SCUIRequestValidationException` is thrown. You can override this default behavior by adding the `scui-suppress-validation-exception` context parameter with the value as `TRUE` in the `web.xml` file located in your `EARFILE/WARFILE/WEB-INF` folder. For example:

```
<context-param>
  <&lt;&lt;<param-name>scui-suppress-validation-exception</param-name>
  &lt;&&&<param-value>TRUE</param-value>
</context-param>
```

When you set this parameter's value as `TRUE`, all the validation exceptions are added to a list that can be accessed using an `ArrayList` as follows:

```
ArrayList<SCUIRequestValidationException>
SCUIWebValidationUtils.getValidationErrorList(HttpServletRequest request)
```

You can also define a global exception handler. If any validation exception has not been detected and goes back to the `SCUISafeRequestFilter`, the request will be sent to the corresponding global error handler servlet container.

This global exception handler and the request method can be defined as context parameters in the `web.xml` file located in your `EARFILE/WARFILE/WEB-INF` folder. For example:

```
<context-param>
  <param-name>scui-global-validation-exception-handler-path
  </param-name>
  <param-value><path_to_global_exception_handler></param-value>
</context-param>
<context-param>
  <param-name><scui-global-validation-exception-handler-method>
  </param-name>
  <param-value>FORWARD|INCLUDE|REDIRECT</param-value>
</context-param>
```

For applications that are based on the Web UI framework, the Sterling Selling and Fulfillment Foundation provides `/jsps/datavalidationerror.jsp` as the default exception handler.

For applications that are based on the HTML UI framework, the Sterling Selling and Fulfillment Foundation provides `/common/datavalidationerror.jsp` as the default exception handler.

The Web UI Framework has also added a Struts Action result "DATAVALIDATIONERROR", which would be returned in case of invalid request. You can define this result type and the corresponding path (say `/jsps/datavalidationerror.jsp`) for these struts actions.

By default, the global exception handler method is set to `FORWARD`.

Defining Error Messages

You can define error messages on each regular expression to show the context sensitive error. The error messages are localizable bundle entries defined in the various bundle file. For example,

```
<RegularExpressions>
  <RegularExpression id="dates" javaPattern="^[a-zA-Z0-9]*$"
    jsPattern="^[a-zA-Z0-9]*$" whitelistErrorMsg="only_alphanumeric_chars_allowed"
    blacklistErrorMsg="alphanumeric_chars_not_allowed"/>
</RegularExpressions>
```

and in the bundle.properties file, add the following entry:

```
only_alphanumeric_chars_allowed={1} must contain only alphanumeric characters.
alphanumeric_chars_not_allowed={1} must not contain any alphanumeric characters.
```

In this case, If the regular expression dates fails in input validation, the error string defined in the bundle keys only_alphanumeric_chars_allowed or alphanumeric_chars_not_allowed is displayed as error message. The name of the input being validated will be used as the second message formatter. The first message formatter is the regular expression.

Note: If whitelistErrorMsg and/or blacklistErrorMsg attribute are not defined, the default error messages are displayed.

Localizing Error Messages

You can localize the error message defined for individual regular expressions.

For HTML UI based applications, the bundle keys for error messages should be present in java bundle files for server side validations and in javascript bundles for client side validations.

For Web UI Framework based applications, the bundle keys for error messages should be present in java bundle files for server side validations and in javascript bundles for client side validations.

For RCP based applications, the bundle keys for error messages should be present in java bundle files for both server and client side validations.

Defining Custom Regular Expression Error Message Provider

This is an optional task. You can provide a custom error message provider by creating an instance of the error message provider class. The error message provider class must implement the ISCRegexErrorMessageProvider interface. Also, the error message provider class must be registered with must be registered with the Sterling Selling and Fulfillment Foundation by adding a new context parameter in the web.xml file ((located inside your EARFILE/WARFILE/WEB-INF directory) as follows:

```
<context-param>
  <param-name><scui-regular-expression-error-message-provider></param-name>
  <param-value><fully-qualified-class-name></param-value>
</context-param>
```

For example,

```
<context-param>
  <param-name>scui-regular-expression-error-message-provider</param-name>
  <param-value>com.text.RegExErrMsgProvider</param-value>
</context-param>
```

Note: Error messages returned by the error message provider class must be bundle keys. The error message will be localized and formatted with regular expression being passed as the first formatter and the input being validated as the second formatter.

Localizing Validation Rules

There cannot be any localization for the validation rules as each rule should validate the input in all available locales. Therefore, you must define the rules that support all the available locales in the rules file itself.

Chapter 7. Building and Deploying Extensions

After You Create Your Extensions

After you are satisfied with all of the extensions you have made to Sterling Selling and Fulfillment Foundation (such as customizing the UI and database, creating custom code and files, and so forth), make sure that you build and deploy the new extensions. You should re-build and deploy the Application Enterprise Archive (EAR) with all the Java files, resource files, JSP files, custom classes, and so forth, that you created or modified.

Building Resource Extensions

To build extensions to application resources, you must rebuild the `resources.jar` file by running the `deployer.sh` (or `deployer.cmd` on Windows) utility from the `INSTALL_DIR/bin` directory. For example:

```
./deployer.sh -t resourcejar
```

This applies to all application resources, including:

- Theme, CSS, Config resources, Data types files, and so forth
- Extended APIs, Events, and XSL templates
- Modifications made in the database, resources, and template directories

To incorporate your JSP or JS file modifications, rebuild the Selling and Fulfillment Foundation EAR.

If you are extending any resource XML files, place your extended `*.xml` files in the `<INSTALL_DIR>/extensions/global/template/resource` folder.

If you are extending any event.xml files, place your extended `*.xml` files in the `<INSTALL_DIR>/extensions/global/template/event` folder.

If you are extending an `*.xsl` file, place your extended `.xsl` files in the `<INSTALL_DIR>/extensions/global/template/xsl` folder. But when providing the name of the template.xsl file during service definition, the path should be `/global/template/xsl/<CUSTOM-TEMPLATE-XSL>`.

Ensure that all the extended JSP and JS files are stored in the `<INSTALL_DIR>/extensions/global/webpages` directory, if they are not already there.

Customizing Resource Bundles

You can define new bundle entries and override out-of-the-box bundle entries.

Server side bundle files are located in the `<INSTALL_DIR>/resources` directory. To change bundle files, add or override the appropriate entries in the `<INSTALL_DIR>/extensions/global/resources/extnbundle.properties` file.

For example, you can add a new entry such as `Detailed_Description=Detailed Description` key-value pair as follows:

1. Add the key-value pair to the `<INSTALL_DIR>/extensions/global/resources/extnbundle.properties` file.
2. Build `resources.jar`.
3. Build the EAR.

Building Other Extensions

About this task

To build your custom code extensions (user exits, extended APIs, custom implementations of supplied Java interfaces, and so forth) modifications, generate a JAR file containing these Java files and custom classes. After creating the JAR file, include the new JAR file in the CLASSPATH environment variable by running the `install3rdParty.sh` (or `install3rdParty.cmd` on Windows) utility from the `<INSTALL_DIR>/bin` directory. For example:

```
./install3rdParty.sh <vendorName> <vendorVersion> <-j | -l | -p | -r | -d >  
<filelist> [-targetJVM DCL | EVERY | NOWHERE | APP | AGENT]
```

Here, `<vendorName>` refers to the name of the vendor such as WebLogic, WebSphere, and JBoss. `<vendorVersion>` refers to the version of the vendor. Pass the appropriate argument based on the file type. You can pass the following arguments:

- `-j` for JAR or compressed files
- `-l` for shared libraries
- `-p` for properties files
- `-r` for resource properties files
- `-d` for database JAR or compressed files

`<filelist>` refers to the path to your custom file.

Note: If you want to make this custom JAR available to the application server and agents when running the `install3rdParty` utility, based on your requirement pass the following target JVM arguments:

- **DCL**—If you want to add the custom JAR to the main dynamic classpath.cfg file only.
- **EVERY**—If you want to add the custom JAR to all the dynamic classpath files (for example, `APPDynamicclasspath.cfg`, `AGENTDynamicclasspath.cfg`, `OPSDynamicclasspath.cfg`, and `ACTIVEMQDynamicclasspath.cfg` files).
- **NOWHERE**—If you just want to add the custom JAR to the `<INSTALL_DIR>/jar` directory and do not want to update any of the dynamic classpath files.
- **AGENT**—If you want to add the custom JAR to the `AGENTDynamicclasspath.cfg` file.
- **APP**—If you want to add the custom JAR to the EAR file.

For example, if you want to add the custom JAR to the `AGENTDynamicclasspath.cfg` file, run the `install3rdparty` command with following arguments:

```
./install3rdParty.sh weblogic 10 -j <Path_to_your_custom_JAR> -targetJVM  
AGENT
```

Note: At times, mechanisms supplied by Sterling Selling and Fulfillment Foundation, such as time-triggered transactions, APIs, and user exits, are replaced by improved mechanisms. When these mechanism are replaced, they are designated as “deprecated.” Whenever possible, use the new mechanisms rather than the deprecated ones. If you do need to use a deprecated mechanism, it must be run in backward compatibility mode. In addition, note that deprecated mechanisms are supported for two major software versions, and then they are removed from the product.

Building Database Extensions

About this task

To build the extensions to your database, re-build the entities.jar by running the deployer.sh (or deployer.cmd on Windows) utility from the <INSTALL_DIR>/bin directory. For example:

```
./deployer.sh -t entitydeployer
```

Note: Before building the database extensions, make sure that all the extension files are stored in the *INSTALL_DIR*/extensions/global/entities directory.

By default, when you run the entitydeployer target, all the log messages are printed to the <INSTALL_DIR>/logs/entitydeployer.log file. If you want to print the log messages in the log file as well as on the console, pass the -l info parameter when you run the entitydeployer target. For example:

```
./deployer.sh -t entitydeployer -l info
```

To update the ERD documentation, re-build the entities.jar by running the deployer.sh (or deployer.cmd on Windows) utility from the *INSTALL_DIR*/bin directory. For example:

```
./deployer.sh -t updateERD
```

Note: By default, when you run the entitydeployer target or InstallService script, the dbverify tool is also run. But if you want to suppress this call to the dbverify tool again when you run InstallService script, override and set the NO_DBVERIFY property to true in the <INSTALL_DIR>/install/properties/sandbox.cfg file. For more information about overriding properties, refer to the Sterling Selling and Fulfillment Foundation: *Properties Guide*.

For deploying the database extensions, refer to *Deploying Extensions*.

Deploying Extensions

After you build the required Sterling Selling and Fulfillment Foundation extensions, you must deploy them.

To deploy the extensions, re-build the EAR file as you did during installation.

IBM recommends that you re-build and deploy the EAR file on your development system and test there first. Then, deploy your extensions to your production system and test them again.

Also, before deploying your extensions on a production system, verify that the `<INSTALL_DIR>/properties/customer_overrides.properties` file has the correct settings. For example, ensure that the cache refresh icons specified in the `yfs.uidev.refreshResources` property is set to N. For additional information about overriding properties using the `customer_overrides.properties` file, see the Sterling Selling and Fulfillment Foundation: *Properties Guide*.

Building and Deploying Enterprise-Level Extensions

You can define resources used by Sterling Selling and Fulfillment Foundation, such as templates, database extensions, UI resources, and so forth, at the Enterprise level. The Enterprise-Level resources are bundled into a services package for deployment.

Enterprise-Level resources can be developed and packaged as an Enterprise service package. This service package contains all of the components required to on-board an Enterprise. The Enterprise-Level resources are identified using a unique resource identifier. The unique resource identifier is used to locate the resources belonging to an Enterprise. Using the unique resource identifier, you can easily deploy or move the Enterprise-Level resources. You define these resource identifiers when you create an organization. For more information about creating an organization, see Sterling Selling and Fulfillment Foundation: *Application Platform Configuration Guide*.

Building Enterprise-Level Extensions

About this task

To build your modified Enterprise-Level extensions such as templates, resource files, customized webpages, entities, and so forth:

Procedure

1. Create a new XML file named as `descriptor.xml` and add the following entry:

```
<ExtensionsDescriptor>
  <<<Package Name="<RESOURCE_IDENTIFIER>" />
</ExtensionsDescriptor>
```

where `<RESOURCE_IDENTIFIER>` refers to the unique resource identifier defined for identifying the resources belonging to an Enterprise.

2. Generate a custom JAR file containing the Enterprise-Level extension files. The custom JAR file should have the following directory structure depending on the extensions you have made:

Note: Before creating the custom JAR, make sure to copy the descriptor `.xml` file to the root of the JAR file.

- `/template/ <TEMPLATE_SPECIFIC_FOLDER>` — for storing the extended template files

Where `TEMPLATE_SPECIFIC_FOLDER` refers to the directory that contains the specific templates. For example:

- `api`—For storing the API-specific templates
- `email`—For storing E-mail-specific templates
- `event`—For storing Event-specific templates
- `userexit`—For storing UE-specific templates

In addition to templates, you can put JAR files, resource files, entity extensions, and customized webpages in the JAR file by creating the required folder in the root of the JAR file. For example,

- /jars —For storing the required JAR files
 - /uijars —For storing the UI-specific JAR files
 - /entities—For storing the extended entity XMLs
 - /webpages—For storing the customized webpages
 - /resources—For storing the modified resource files
3. After creating the JAR file, deploy the new JAR file by running the InstallExtensions.sh (or InstallExtensions.cmd on Windows) utility from the *INSTALL_DIR/bin* directory. For example:

```
./InstallExtensions.sh <filename>
```

Here, <filename> refers to the path to the JAR file you created in step 2.

What to do next

After building the template extensions, make sure that you re-build the resources.jar.

Building Enterprise-Level Resources Extensions

About this task

To build your modified Enterprise level UI resource files (such as theme, css, config resources, or datatype maps files), re-build the resources.jar file by running the deployer.sh (or deployer.cmd on Windows) utility from the *INSTALL_DIR/bin* directory. For example:

```
./deployer.sh -t resourcejar
```

To incorporate your JSP or JS file modifications, re-build the Sterling Selling and Fulfillment Foundation EAR.

Note: Make sure that all of the extended JSP and JS files are stored in the *INSTALL_DIR/repository/eardata/smcfs/war* directory.

Building Enterprise-Level Database Extensions

About this task

To build your modified Enterprise-Level database, re-build the entities.jar file by running the deployer.sh (or deployer.cmd on Windows) utility from the *INSTALL_DIR/bin* directory. For example:

```
./deployer.sh -t entitydeployer
```

Note: Before building the database extensions, make sure that all of the extension files are stored in the *INSTALL_DIR/repository/entity/extensions* directory.

Note: By default, when you run the entitydeployer target or InstallService script, the dbverify tool is also run. But if you want to suppress this call to the dbverify tool again when you run InstallService script, override and set the NO_DBVERIFY property to true in the *INSTALL_DIR/install/properties/sandbox.cfg* file. For more information about overriding properties, refer to the Sterling Selling and Fulfillment Foundation: *Properties Guide*.

By default, when you run the entitydeployer target, all the log messages are printed to the `INSTALL_DIR/logs/entitydeployer.log` file. If you want to print the log messages in the log file as well as on the console, pass the `-l info` parameter when you run the entitydeployer target. For example:

```
./deployer.sh -t entitydeployer -l info
```

Building Enterprise-Level Template Extensions

The system automatically reads the extended templates from the template folder that gets created in the `INSTALL_DIR/extensions` directory after you run the `InstallExtensions.sh` script.

Deploying Enterprise-Level Extensions

About this task

After you build the required Enterprise-Level extensions, you must deploy these extensions.

To deploy the Enterprise-Level extensions, re-build and deploy the Sterling Selling and Fulfillment Foundation Enterprise Archive (EAR). For more information on how to build an EAR, see *Sterling Selling and Fulfillment Foundation: Installation Guide*.

Note: IBM recommends that you re-build and deploy the EAR file on your development system and test there first. Then, deploy your extensions to your production system and test them again. For information about deploying Sterling Selling and Fulfillment Foundation, see *Sterling Selling and Fulfillment Foundation: Installation Guide*.

Also, before deploying your extensions on a production system, ensure that the `INSTALL_DIR/properties/customer_overrides.properties` file has the correct settings. For example, ensure that the cache refresh icons specified in the `yfs.uidev.refreshResources` property is set to `N`. For additional information about overriding properties using the `customer_overrides.properties` file, see the *Sterling Selling and Fulfillment Foundation: Properties Guide*.

Customizing Web.xml

Customizing web.xml for Multiple Applications

Sterling Selling and Fulfillment Foundation provides the following extensions for `web.xml`.

The extension package should have a ".extn" suffix and the "package_name" attribute should be used to specify the package. For example:

```
<WebComponents Package = "package_name.extn">
```

Applications have the capability to suppress some common configurations. Applications can suppress these configurations by using the `Suppress` element. The suppression will be done by removing any element that matches the suppression criteria. For example:

```
<WebComponents Package = "package_name.extn">  
<Suppress>  
<servlet><servlet-name>JasperPDFReport</servlet-name></servlet>  
servlet-mapping<<servlet-name>JasperPDFReport</servlet-name>
```

```

</servlet-mapping>
</Suppress>
<web-app>
<!-- All the web.xml pieces needed, in standard web.xml format. -->
</web-app>
</WebComponents>

```

In this example, all configurations are suppressed in which the servlet element contains a child `servlet-name` with the given name.

Customizing web.xml for Session Timeouts

The default session timeout value is 6000 seconds set from the `SessionTimeout` value in the `YFS_USER` table. To set a different session timeout, configure two parameters in your `web.xml` file:

- A context parameter to allow the timeout value to be set from the file

Note: If this value is not set, the session timeout parameter is ignored.

- A session timeout parameter to set the numeric value

To customize session timeouts:

1. Edit your `EARFILE/WARFILE/WEB-INF/web.xml` file to add the context parameter `scui-suppress-user-level-sessiontimeout-override`. Set the value to `y`:

```

<context-param>
  <param-name>scui-suppress-user-level-sessiontimeout-override</param-name>
  <param-value>y</param-value>
</context-param>

```

This allows the session timeout to be set from the timeout value.

2. Add an entry to `web.xml` to set the session timeout configuration parameter, in minutes:

```

<session-config>
<session-timeout><timeout_value_in_minutes></session-timeout>
</session-config>

```

Deploying the Enterprise Archive Package

About this task

To make Sterling Selling and Fulfillment Foundation available for use, you must create and deploy the application EAR file.

Procedure

1. Set up the application server appropriately for deploying the application. For more information about setting up the application server, refer to the installation documentation.
2. Create the EAR package for the application server.
 - To create the application EAR file for a single WAR deployment, run the following command from the `<INSTALL_DIR>/bin` directory: UNIX

```
./buildear.sh (.cmd for Windows) -Dappserver=<application server> -Dwarfiles=<war file> -Dearfile=<ear file>
```
 - To create the application EAR file for a multiple WAR deployment, run the following command from the `<INSTALL_DIR>/bin` directory:

```
./buildear.sh (.cmd for Windows) -Dsupport.multi.war=true
-Dappserver=<application server> -Dwarfiles=<war file,<comma-separated packages>
-Dearfile=<ear file>
```

To create additional WAR files, add the appropriate packages to the value of the `-Dwarfiles` argument, separated by commas. For example, to create three WAR files, set the `-Dwarfiles` argument in the commands in Step 2 as follows:

```
-Dwarfiles=<war file 1>,<war file 2>,<war file 3>
```

Running the command in this step creates the EAR file in the `<INSTALL_DIR>/external_deployments` directory. It also places the multiple war files in the EAR file.

3. Deploy the EAR file on the application server. For more information about creating and deploying the EAR file, refer to the installation documentation.

Deploying Multiple EARs on One Application Server

Sterling Selling and Fulfillment Foundation provides support for deployment of Multiple EARs (Enterprise Archives) on single application server. On the same application server, you can do either of the following:

- Deploy different customizations of the same or different versions of the application.
- Deploy different versions of the same application.

The number of different EARs that can be deployed on a single application server depends on the available resources on the application server. To support this deployment of multiple EARs, the different versions or customizations of the same application should be deployed as an EAR and not in exploded mode.

Note: Sterling Selling and Fulfillment Foundation assumes that each EAR file is generated from a different `<INSTALL_DIR>` directory.

To deploy multiple EARs, you must do the following:

- Define JNDI (Java Naming and Directory Interface) Context Namespace
- Define Context Root Entries

Defining the JNDI Context Namespace

About this task

You must define the JNDI (Java Naming and Directory Interface) context namespace property to avoid JNDI clashes. To define the JNDI entries, do the following:

Procedure

1. Edit the `<INSTALL_DIR>/properties/sandbox.cfg` file.
2. Add the `YFS_CONTEXT_NAMESPACE` property and assign it a name.
3. Run the following script from the `<INSTALL_DIR>/bin` directory to munge the value of this property to the `yifclient.properties` file:
 - `setupfiles.sh` (for UNIX/Linux)
 - `setupfiles.cmd` (for Windows)

Defining Context Root Entries

About this task

You must add the WAR file mappings for defining the context root in the `build.properties.in` file. This ensures that the web applications that are installed on single application server have unique context roots.

Note: Multiple EARs or context roots require additional memory for the application server JVM. Testing has shown that the deployment of a second application EAR file requires 2.5 - 3.5 times the memory of a single EAR. Supporting two deployments may require up to 2.5 GB of heap space and 1.2 GB of permanent space.

During installation, you can use JVM-specific arguments to avoid out-of-memory errors. For more information, see the Sterling Selling and Fulfillment Foundation: *Properties Guide* descriptions of `ADDITIONAL_ANT_JAVA_TASK_ARGS` and `ADDITIONAL_ANT_COMPILER_TASK_ARGS`.

To add these entries, do the following:

Procedure

1. Edit the `<INSTALL_DIR>/install/bin/build.properties.in` file.
2. Add the WAR file mappings for the context root paths for any web application.

The key should be the name of the application's WAR file. For example,

```
platformdemo.war=/myplatformdemo
yantrawebsservices.war=/yantrawebsservices
platformdemodocs.war=/myplatformdemodocs
```

If you want to deploy the same WAR file twice, use WAR file mappings like the following example:

```
platformdemo1.war=platformdemo,platform
platformdemo2.war=platformdemo,platform
```

Then, during the EAR build, pass the following argument:

```
-Dwarfiles=platformdemo1,platformdemo2
```

3. Run the following script from the `INSTALL_DIR/bin` directory to munge the context root WAR file mappings to the `build.properties` file:
 - `setupfiles.sh` (for UNIX/Linux)
 - `setupfiles.cmd` (for Windows)

Results

Note: If you provide `yfs.context.namespace` property and don't provide the mapping of the WAR files in the `build.properties.in` file, the `buildEAR.cmd` (`buildear.sh` for UNIX/Linux) script will throw an error forcing the user to fill in the WAR file mappings. And if you provide the mapping of the WAR files in the `build.properties.in` file and don't provide `yfs.context.namespace` property, the WAR file mappings are ignored.

Also, the `yfs.context.namespace` property value and the WAR file mappings name should be different for each EAR that you want to deploy on the same application server.

Chapter 8. File Names, Keywords, and Other Conventions

Reserved Special Characters and Keywords Introduction

Sterling Selling and Fulfillment Foundation reserves keywords and special characters that are only used internally. For more details about using special characters, see the Sterling Selling and Fulfillment Foundation: *Customizing APIs Guide*.

Naming Files

When naming files, IBM recommends that you choose characters from the standard English-based character set, such as A through Z and 0 (zero) through 9. That way, if you need to localize the application in languages other than English, you do not need to rename files.

Reserved Keywords

Some keywords are reserved for use by the Sterling Selling and Fulfillment Foundation and are important to keep in mind when programming with APIs and creating error codes. Do not create file names or error codes that start with the following:

- DCS
- INV
- OMD
- OMP
- OMR
- OMS
- PLT
- SYS
- WMS
- YCM
- YCP
- YCS
- YDM
- YFC
- YFE
- YFS
- YFX
- YIC
- YIF
- YPM
- YRET

Using Multi-Byte Characters

If you want to use multi-byte characters, your database must be configured to support multi-byte characters. For more information about multi-byte characters and localization, see the Sterling Selling and Fulfillment Foundation: *Localization Guide*.

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](http://www.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:

Printed in USA