Gentran Integration Suite™

MESA Developer Studio

Version 4.2

Sterling Commerce
An IBM Company

Contents

Gentran Integration Suite Architecture 25

Reference Information for Developing a Service 37

About MESA Developer Studio

MESA^{TM} Developer Studio is used to create and edit Gentran Integration Suite property files and business processes. You can also use MESA Developer Studio to remotely start and stop a Gentran Integration Suite instance, install third party files, list directory contents and current processes, and display disk usage. For example, within Gentran Integration Suite you can only edit business processes using the Graphical Process Modeler (GPM). From MESA Developer Studio, you can also edit business processes using a code editor.

Note: Before you can begin using MESA Developer Studio, you must first install MESA Developer Studio. For more information, see the Gentran Integration Suite *Installation Guide*.

Using MESA Developer Studio to Manage Gentran Integration Suite

Caution: MESA Developer Studio is designed to assist you with Gentran Integration Suite resource development. Changes made with the MESA Developer Studio plug-ins should be thoroughly tested in a development environment before moving them into production.

Use MESA Developer Studio to:

- Edit Gentran Integration Suite property files
- Work with business processes
- Start and stop Gentran Integration Suite instances
- Install third-party files
- Manage Gentran Integration Suite resources
- List directory contents
- List current processes
- Display disk usage

Available MESA Developer Studio Editors

The following editors are available in MESA Developer Studio to assist you in creating or editing properties, service definitions, and other code:

- Properties Editor
- JDBC Properties Editor
- Knowledgebase Properties Editor
- Service Definitions Editor
- BPML Editor

License Management Settings

Licenses provide you access to the different components offered by Gentran Integration Suite that you have purchased. Without the proper licenses, Gentran Integration Suite does not operate. For example, after you purchase Gentran Integration Suite, you can purchase new components and open those components with a new license file. Occasionally, you may need to update your license file for either administrative purposes, or when your license file expires. You can view the components license status and update your license files through MESA Developer Studio.

To manage your license files:

1. From the MESA Developer Studio perspective, double-click an instance.
   The instance overview appears in the Control Editor.
2. Click the Settings tab at the bottom of the Control Editor pane.
3. Click **Update Licenses**.
4. Navigate to the appropriate license file. Click **Open**.
   The license file is automatically updated.

Creating a MESA Developer Studio Project

Create a project in the Package Explorer to organize the files and resources you will use on your local system.

To create a project:

1. From the File menu, select **New > Project**.
2. Select **Simple > Project**.
3. Click **Next**.
4. Type a name for the project.
5. Click **Finish**. A new project is added to the Package Explorer under the folder name Other Projects.

Note: If no direct connection is possible between the host where Gentran Integration Suite is installed and the Windows PC where Eclipse is installed, and you are using a proxy server, you must enable the HTTP proxy connection. From the Window menu, select **Preferences**. On the left, select Install/Update. In the Proxy settings section, add your proxy information and click **Apply**.

Managing Resources in MESA Developer Studio

Resources are files, templates, and documents that may be deployed in Gentran Integration Suite and that you can import and export from one Gentran Integration Suite system to another, such as when you are migrating from a test to production environment. You can check in, check out, lock, and unlock the following resources in MESA Developer Studio:

- Business Processes – Business process model definitions and their associated particulars.
- XML schemas – Data that makes up XML schemas.

Working with Business Processes

Business processes may be created and edited in MESA Developer Studio using either the BPML text editor, or the GPM. Files with a .bp extension will by default open the GPM; however, they may also be edited by using a option to open the file with the BPML Editor.

Note: The default when you double-click a business process is for it to open in the GPM. If you want to edit a business process using the BPML Editor, right-click on the business process and select **Open With > BPML Editor**. MESA Developer Studio remembers this setting so that the next time you double-click on that business process, it will open in the BPML Editor.

The BPML Editor allows XML-like text editing of business processes along with autofill of BPML activities. MESA Developer Studio allows you to start the GPM; however, it runs independently of Eclipse.

Setting Up the GPM

To use MESA Developer Studio to edit business processes, you must set up the GPM to be used in MESA Developer Studio. This enables you to launch the GPM from within Eclipse when you double-click on a business process name, either in your Package Explorer list of projects (local) or on your test server through a configured instance. (Only files that have been checked out from the server are displayed in that view.)

To set up the GPM:

1. In Eclipse, from the Window menu, select **Preferences**.
2. On the left, select the **Java > Installed JREs** category.
3. In the Installed JREs section, click **Add**.
4. Enter the following information:
   - JRE name – Type **GBM JRE**.
   - JRE home directory – Click **Browse** to select the directory. You must use a 1.5 JRE with the GPM. This is usually located in C:\Program Files\javasoft\jre\1.5.
5. Click **OK**. You are now ready to edit business process files from within MESA Developer Studio.

Note: The first time you start the GPM from Eclipse, your Gentran Integration Suite instance must be running.

For more information on managing business processes, see the business process documentation.

Working with Schemas

An *XML schema* is an XML document that specifies the structure of a valid XML document. Comparable to a document template, an XML schema ensures that every item is in the correct form. An XML schema consists of the following components:

- XML declaration – Defines the XML version that the schema uses
- Schema element – Identifies the document as an XML schema
- Element declaration – Defines the element
- Attribute declaration – Defines the attribute

Schemas are used in Gentran Integration Suite to validate translation data in the Map Editor. When you submit an XML document using Gentran Integration Suite, the XML document is compared to the XML Schema to ensure that the document is in the appropriate format and is valid. For more information on managing schemas, see the schema documentation.

Working with Properties Files

From Mesa Developer Studio, you can add, edit, or delete properties within existing files.

To open a property file:

1. From the MESA Developer Studio perspective, expand the Properties Files folder.
2. Double-click on the property file name you want to work with to open it in the editor.
3. Do one of the following:
    - Click **Add** to add a property. Type a name and value for the property.
    - Select a property and click **Delete** to remove it.

Creating Services Using MESA Developer Studio SDK

Creating Custom Services

Gentran Integration Suite can perform most common tasks needed by a user, but there are instances where functionality is needed that is not provided and an existing service is not available. This usually occurs in environments with legacy systems that do not use standards-based methods for communication. There are several ways to interact with these systems, including the Command Line adapter and the Scripting adapter, but the capabilities of these adapters are limited.

MESA^{TM} Developer Studio Software Development Kit (SDK) provides the tightest integration with Gentran Integration Suite and enables you to create complex and complete services and adapters. These services use the same APIs as the services included with Gentran Integration Suite, so all of the benefits and infrastructure of Gentran Integration Suite are available. This flexibility and tight integration means that creating a service is more advanced and requires good knowledge of Java development, the Gentran Integration Suite APIs, and the APIs of any system that will be accessed by the service.

Note: Because all adapters are a type of service, this guide uses the term *service* for both services and adapters. This guide uses the term *adapter* when the information is unique to adapters.

Anatomy of a Gentran Integration Suite Service

This section describes and explains the various service components that Gentran Integration Suite uses.

What is a Service?

A *service* is a component that can be configured to carry out an activity in a Gentran Integration Suite business process.

What is a Parameter?

Parameters can be configured to define and control your service. Any parameters that you want to add to a service must be added to a parameter group.

What is a Parameter Group?

Parameter groups are logical groupings of similar parameters (for example, host name and port). It is acceptable to have a parameter group with only one parameter. There are three types of parameter groups:

- Global
- Instance
- Workflow

The following table describes the types of parameter groups. The location refers to the area in Gentran Integration Suite where this service parameter may be edited in addition to MESA Developer Studio SDK:

Parameter Group	Description	Location for End User
Global Definition	Widest possible scope, applicable to all services of this type. They have a constant value for all instances of a service.	Gentran Integration Suite interface: Deployment > Services > Installation/Setup.
Workflow Definition	Specific to a single invocation of a service. May have different values every time the service is called.	Graphical Process Modeler.
Instance Definition	Specific to a single copy of a service. May have different values for each instance of a business process that calls the service instance.	Page in a service configuration wizard accessed through Gentran Integration Suite interface, Deployment > Services > Configuration.

What is an Adapter?

An *adapter* is a type of service that communicates with external systems to move data in and out of Gentran Integration Suite.

What is a Method?

A *method* is the Java equivalent of functions, subroutines, or procedures in other programming languages.

About MESA Developer Studio SDK

The MESA Developer Studio SDK helps you create and edit custom services and adapters for Gentran Integration Suite using the Eclipse development environment. The MESA Developer Studio SDK is designed as an Eclipse plug-in, installed locally, that extends the Eclipse Integrated Development Environment (IDE). Use the SDK to create a service, build and export a service package within the Eclipse development environment, then install and test it with Gentran Integration Suite.

Like the MESA Developer Studio plug-in, the SDK runs independently from Gentran Integration Suite in the Eclipse IDE.

Upgrading from Previous Versions

MESA Developer Studio SDK is available for use with Gentran Integration Suite. You cannot open projects created with the previous Service SDK in MESA Developer Studio SDK; however, you can import your existing java files from an old project into a new SDK project. MESA Developer Studio SDK includes all of the previously available Service SDK features, and includes new and enhanced features such as code editors, validation, consistency check, and wizards that guide you through specific tasks.

Directory Structure for MESA Developer Studio SDK Projects

MESA Developer Studio SDK creates a directory structure in the location you specify when you create a new service project (servicename.xml). The following table describes the files the SDK creates in the location you specified:

Directory Path	File	Contents
src/<packagename>	serviceImpl.java	Service Harness Implementation portion of the adapter. Contains the required processData() function for interaction with the activity engine.
src/<packagename>	servicenameServer.java	RMI remote interface portion of the adapter that describes the functions that can be called. This file is created for adapters only.
src/<packagename>	servicenameServerImpl.java	RMI implementation portion of the adapter. This file is created for adapters only.
src/<packagename>	ServicenameResources.properties	Property file that holds strings mapped to certain messages. Primarily used for logging purposes.
servicedefs	servicename.xml	How the user interface is presented by the configuration wizard at run time.
ui/properties/lang/<*LangAbrev*>	servicename_*LangAbrev*.properties	Language properties in the default language of the computer on which you are working. For example, servicename_en.properties is an English language property file. For more information about reference information, see *Reference Information for Developing a Service* on page 37.
ServiceSDK Libraries	woodstock.jar Security.jar	Gentran Integration Suite related libraries used by the SDK during compilation.

Optional Directories Within a Service

The following table lists the optional directories and their corresponding files in alphabetical order:

Directory Path	File	Contents
bpml/<subdir>	*.bpml	Business processes that belong to this service. **Caution**: Use a subdirectory within bpml for your bpml files. If you do not use a subdirectory for your bpml files, they will not be installed correctly when your service package is deployed.
db	*.dat	Database specific files that belong to this service.
files	*	Additional files that belong to this service, optionally organized in subdirs.
import	*.xml	Gentran Integration Suite Resources to be imported that belong to this service.
scripts	PostServiceInstall.class PreServiceInstall.class	Pre-/Post Installation java classes that belong to this service.
schema	*.dtd, *.xsd	XML Schema files that belong to this service.

Note: Do not create optional directories that do not contain at least one file. Empty directories will cause the deployment of a service to fail.

Using the MESA Developer Studio SDK Cheat Sheet

MESA Developer Studio provides a cheat sheet to guide you through the service development process. The SDK Cheat Sheet provides you with information and step-by-step help to create a service by listing the sequence of steps required to create and package a service. As you progress from one step to the next, the cheat sheet automatically launches the required tools for you. If there is a manual step in the process, the step will tell you to perform the task and click a button in the cheat sheet to move on to the next step. Relevant help information is also available to guide you.

To access the SDK Cheat Sheet:

1. Open the MESA Developer Studio SDK perspective.
2. From the Help menu, select **Cheat Sheets**.
3. In the Cheat Sheet Selection window, expand the Gentran Integration Suite Studio folder and select **MESA Developer Studio SDK**.
4. Click **OK**. The MESA Developer Studio SDK Cheat Sheet opens on the right.

Steps to Create a Service Using MESA Developer Studio SDK

The process of creating and installing a service using MESA Developer Studio SDK involves a series of several steps. The following steps provide a high-level overview of what is required. Click a corresponding link to see detailed instructions for each step.

To create and install a service:

1. Start MESA Developer Studio SDK. See *Starting MESA Developer Studio SDK* on page 14.
2. Create a new SDK project. See *Starting MESA Developer Studio SDK* on page 14.
3. Add business logic. See *Adding Business Logic to a Service* on page 16.
4. Add service parameters (optional). See *Adding Parameters to the Service Definition File* on page 16.
5. Add any additional objects (optional). See *Optional Directories Within a Service* on page 13.
6. Build a service package. See *Exporting a Service for Deployment* on page 20.
7. Install and run the service in a Gentran Integration Suite test instance and verify that the service works as expected. See *Installing a Service into Gentran Integration Suite* on page 21.
8. Install the service to the Gentran Integration Suite production environment.

Starting MESA Developer Studio SDK

Start MESA Developer Studio SDK from your computer by launching Eclipse. The SDK is run independently from Gentran Integration Suite. The WebDAV server must be running the first time you launch the SDK and each time you want to deploy a service package to the instance.

Note: If you are creating adapters, verify that the system with which you will be integrating Gentran Integration Suite is implemented and working correctly.

To start MESA Developer Studio SDK:

1. Launch Eclipse.
2. From the Window menu, select **Open Perspective** > **Other**.
3. From the list, select **MESA Developer Studio SDK**.
4. If this is the first time you have launched SDK, you are asked to enter license information.
5. Type your WebDAV info, and user ID/password from Gentran Integration Suite.

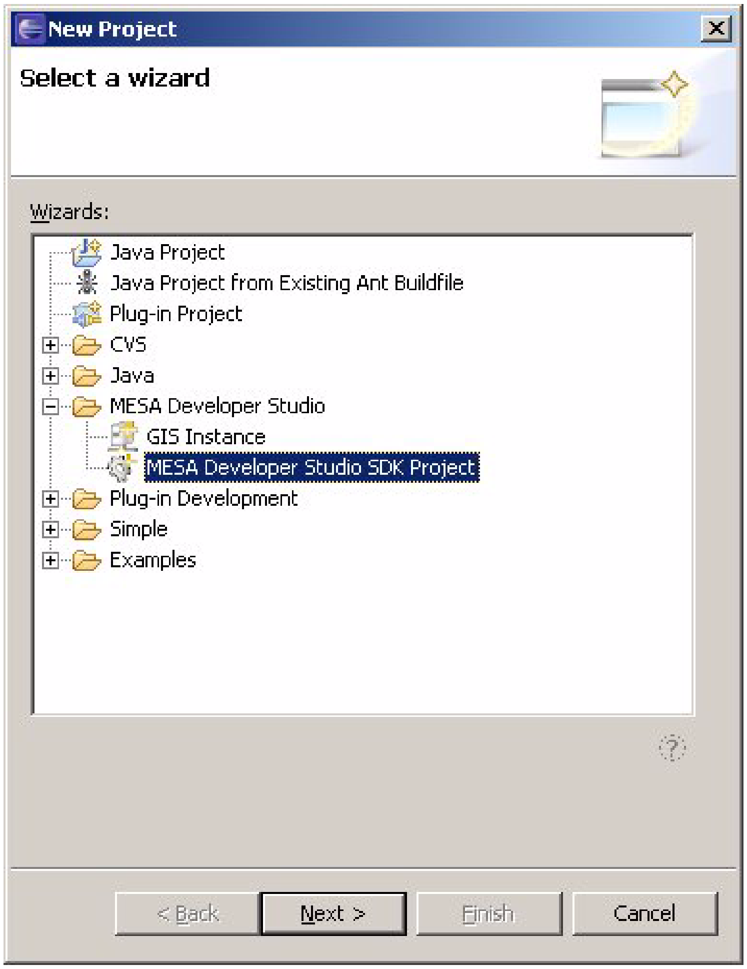
Creating a MESA Developer Studio SDK Project

Caution: Services developed with the SDK should be tested and deployed in a test environment before being deployed to a production instance.

You can either use the Cheat Sheet (see *Using the MESA Developer Studio SDK Cheat Sheet* on page 13) or follow these steps:

To create a project:

1. From the File menu, select **New** > **Project**.
2. Select **MESA Developer Studio** > MESA Developer Studio **SDK Project** and click **Next**.



3. Type a project name, which is a unique name for the project, and click **Next**. Do not use spaces.
4. Complete the following Service Profile information and click **Next**:
   - Service name – unique name for the service, using Java naming standards. Defaults to project name.
   - Service package - name of the Java package where the service should be stored. Defaults to com.mypackage.
   - Service label – name of the service as it should appear in Gentran Integration Suite UI. Defaults to project name. You should make this name unique so that it can be more easily recognized in Gentran Integration Suite.
   - Service description – meaningful description as it should appear in Gentran Integration Suite UI. Defaults to start with “This service implements…”
   - Service Version – required for the service definition file. System created .

- Service type - Service or Adapter. If you select adapter, also select if it should be stateless. For more information, see *Stateless and Stateful Adapters* on page 40.

5. Complete the Build Options and click **Next**:
   - Document Storage Type Options – How the service running in Gentran Integration Suite will store documents. This option is used in the Big A and Little a.
   - Code generation options – if checked the selected Document Storage Type option will be used.
   - Create project folder options – See *Optional Directories Within a Service* on page 13 for more information.
6. Select the Gentran Integration Suite Library version. The Gentran Integration Suite library version available depends on the instance you are connecting to. You can load additional libraries. See *Changing the SDK Library Version* on page 20.
7. Click **Finish**. The project is created. The system creates all required fields that represent a deployable service.

   You should now edit the resource and Java files as needed to develop the service. For more information, see *Adding Business Logic to a Service* on page 16.

Note: Saving the project regenerates the view of the project. The project should be saved anytime a change is made that affects the navigation options for the project. For example, adding a new file or folder creates a new navigation object.

Adding Business Logic to a Service

You must add business logic to the service. This is what makes the service perform the intended tasks. In this step you extend the generated Service code by your own business logic to the Big A portion of the service using Java:

1. From the Window menu, select **Show Views > Other > Basic > Tasks**.
2. Double-click on a TODO item. In the Eclipse editor, you should see the method processData().
3. Add logic that fits your service or adapter.
4. Save the project and regenerate the code.

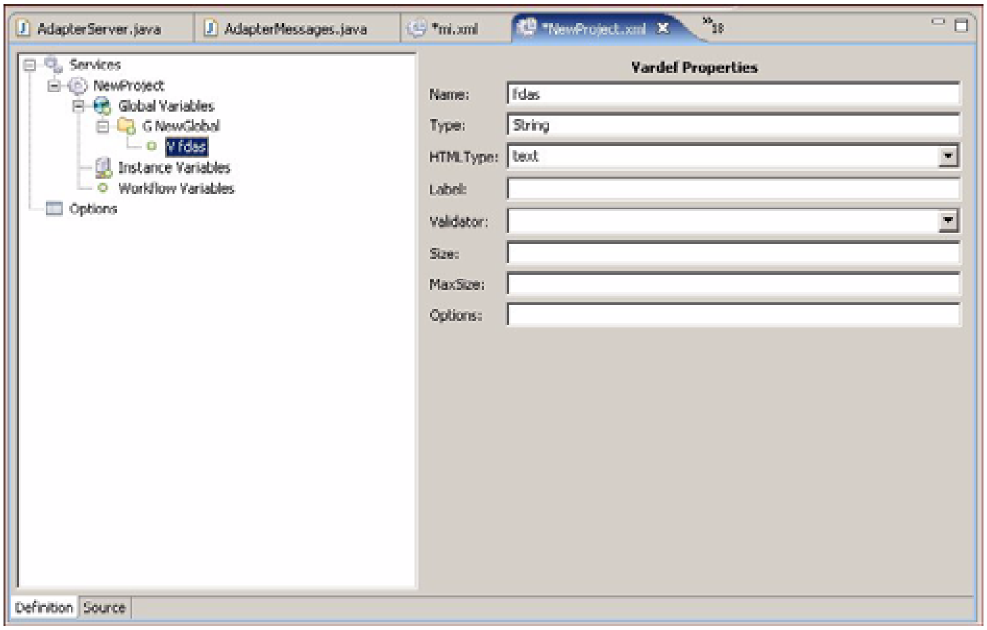
Adding Parameters to the Service Definition File

In this step you can define service parameters that can be used to configure and control your Service.

To add a parameter to a service:

1. In the Package Explorer, expand the **servicedefs** node and right-click on **<service name>.xml**.
2. Select **Open With > Service Definition Editor**. The following view opens with the parameter group types displayed.
3. Right click on a parameter group type and select **New Group**.
4. Type a title for the new group and click **OK**.
5. Click the new group title to add instructions to the group properties.

6. Add parameters to the group. Right-click on the group title and select **New Vardef**.
7. Type a name for the vardef (variable definition) and click **OK**.
8. Click the new vardef to add properties to this parameter.



Vardef Properties

Name: fdas

Type: String

HTML Type: text

Label:

Validator:

Size:

MaxSize:

Options:

9. Complete the following:
   - Name – Required. Name of the parameter as it will appear to the user. System provided. Cannot contain spaces.
   - Type – Required. Java type of the parameter. Default value is string.
   - HTML Type – Required. HTML input type of the parameter. Valid values are: Text, Select, and Radio. Default is Text
   - Label – Required. Cannot contain spaces.
   - Validator – Optional. Type of validator. Select from the list.
   - Size – Optional. Number of characters for the parameter display size.
   - Max Size – Optional. Maximum number of characters allowed for the parameter.
   - Options – Optional.
10. Click **File** > **Save project**.
11. The Language property file (ui/properties/lang/en/<service name>_en.properties) contains label/value pairs that allow to give labels (for example, variables) a descriptive name in the user interface. If the language property file does not contain a label for each corresponding entry you will receive an error message. Right-click on the error message and select **Quick Fix**.
12. Select the desired fix and click **OK**. The language property file is updated with the new label and an editor with the updated language property file displays.
13. Save the language property file.

Note: Each time a Service Definition file or the language property file is saved, a consistency check between both files is performed.

Adding Resources to a Service

Depending on the service you are creating, you may need to add third-party files such as BPML, scripts, databases, properties, libraries, and .jar files. In addition to adding these resources, you can also remove resources from the project file (the original files are not deleted from their original location). Additionally, you can create folders for any files you want to add to the service project.

The Service SDK adds these files in the location you specify when you create the service. If you add any of these resources, you must add the folders described in the table of optional directories. For more information, see *Optional Directories Within a Service* on page 13.

Writing Log Messages into the Gentran Integration Suite Message Log File

MESA Developer Studio SDK supports the user in externalizing log messages strings into separate message property files which can be used in the Java classes of the service. Service Projects contain the following java classes for defining Log messages:

- Message String Declaration File: `<service_name>Messages`
- Message Property File : `<service_name>Resources.properties`

To write log messages:

1. Edit declaration file <service_name>Messages. Declare a constant String variable for each log message you want to use. The declaration has following Syntax: public static String <MessageID>. It is useful to start the message ID with a component prefix.

   Example: `public static String ExampleImpl_NoTicketsAvail`
2. Edit resource file <service_name>Resources.properties. The log message entries have following simple Syntax: <MessageID>=<Message_Text>.

   Example: `ExampleImpl_NoTicketsAvail=No more tickets available`
3. Use Log Message in Java Source Code Editor. To write a error message with the XLogger "log" the method "logError" is used which takes a "String" parameter as an argument. To pass the name of the message string simply write "ExampleMessages" then use short cut "ctrl-tab" to open the Eclipse Java Editor auto-complete drop-down box, select the log message ID and hit Return.

   Example: `log.logError(ExampleMessages.ExampleImpl_NoTicketsAvail)`

Creating a serviceinstances.xml File

To create a Service instance:

1. Right-click on the project name and select **New > File**.
2. In the File Name field, type **serviceinstances.xml**.

3. Click **Finish**. A new file is created in the project.
4. Open the file serviceinstances.xml in an XML editor and define your adapter instance as described in the previous example.
5. If there is already a serviceinstances.xml file, you can import that file to your project. When you deploy your adapter package the adapter instance is created automatically.

Example (Serviceinstances.xml)

In the following example for an serviceinstance.xml file an Adapter instance MyExample is created for the Adapter Example which is described in the service definition file below. The Adapter Example has only one instance variable UserName. In the Adapter Instance MyExample, the instance variable UserName is configured with value Smith.

```
<?xml version="1.0" encoding="UTF-8"?>
<services>
<service parentdefname="ExampleAdapter"
name="MyExample"
      displayname="Example Adapter"
      description="Test Instance of ExampleAdapter"
targetenv="all"
activestatus="1"
systemservice="0"
parentdefid="-1">
<parm name="UserName" value="Smith"/>
</service>
</services>
```

Service Definition File

```
<SERVICES>
  <SERVICE name=" ExampleAdapter "
           description="example.description"
           label="example.label"
           implementationType="CLASS"
   JNDIName="com.mycompany.example.ExampleAdapter"
           type="Adapter"
           adapterType="STATEFUL"
            adapterClass=" com.mycompany.example.ExampleAdapterImpl"
            version="3.0"
            SystemService="NO">
    <VARS type="instance">
 <GROUP title="example.group1.title"
   instructions="example.group1.instructions">

          <VARDEF varname="UserName" type="String" htmlType="text"
                  validator=" ALPHANUMERIC " size="20" maxsize="40"
                  label="example.username" />
       </GROUP>
    </VARS>
  </SERVICE>

</SERVICES>
```

Changing the SDK Library Version

If you need to use SDK Libraries other than those supplied with your version of Gentran Integration Suite (woodstock.jar and Security.jar), you can switch to a different (older or newer) Gentran Integration Suite SDK Library.

To change library versions:

1. Download the required two jar files to the Windows machine where Eclipse and MESA Developer Studio SDK are installed.
2. Navigate to the directory of your Eclipse installation to the <Eclipse-root>\plugins\com.sterlingcommerce.mesa.servicesdk_1.0.0\lib directory. The directory 4.1.0-0 that contains the two SDK Libraries (jar files) delivered with this version are listed.
3. On the same level, create a subdirectory containing the name of the additional or new Gentran Integration Suite version and copy the two jars from step 1 into the new directory.
4. Navigate to the <Eclipse?root>\plugins\com.sterlingcommerce.mesa.servicesdk_1.0.0\res directory. This directory holds a subdirectory with the Gentran Integration Suite version (same name as in step 2) that contains a file AntExport.xml which is used by the Export Wizard.
5. Similar to step 3, create in the <Eclipse?root>\plugins\com.sterlingcommerce.mesa.servicesdk_1.0.0\res directory a new subdirectory with the name of the other Gentran Integration Suite version (for example, 4.1.0-1).
6. If you did not create a newer AntExport.xml file, copy the file from Gentran Integration Suite version 4.1.0-0 to the new directory. If you have a newer AntExport.xml file you may copy this one to the new directory.
7. Repeat steps 1 - 6 for all different versions of SDK Libraries you require in the SDK.
8. Start Eclipse and switch the Perspective to MESA Developer Studio SDK.
9. If you want to change the Gentran Integration Suite version of a service that is already created, select SDK Libraries [4.1.0-0] from your project the directory.
10. Right-click and select **Configure**.
11. Select from the drop down any of the SDK Libraries you have placed according to steps 1 -6.
12. Click **Finish**. The new SDK library version is now available from the list in the New Project Wizard.

Exporting a Service for Deployment

Once you have created a service you can package it for installation into Gentran Integration Suite. In this step you package all required service resources of your project in a package (Jar-archive) that can be deployed into a Gentran Integration Suite system.

To build a service package:

1. Select the project you want to export in the Package Explorer. You can package more than one service at the same time by using the Ctrl key when selecting.
2. Right-click and select **Export**.

3. In the Export window, select Service Package as the export destination and click **Next**.
4. Browse to select the destination directory.
5. Click **Finish**. The service package <service name>_<version>.jar is built and placed in the <selected package folder>/<service name>/dist/<service name> folder. You may be prompted to save resources before the export is executed.

Note: The export process always exports the entire project even if you selected only one or more of its subcomponents.

The export process writes to the AntExport.log file in the <destination directory/adaptername> directory for with the results of the packaging process. The service is now ready to be installed into Gentran Integration Suite.

Installing a Service into Gentran Integration Suite

After you create a service and package the source code, you must install the service package into Gentran Integration Suite.

Note: Before you install the service package into a Gentran Integration Suite production environment, you should install and test it in a test environment.

To install a service package:

1. From the MESA Studio perspective, choose the instance in which you want to install the package.
2. Right-click and choose **Install Service Package**.
3. Locate the package file. Click **Open**.
4. Click **Finish** and restart the instance.

Working with the MESA Developer Studio Skin Editor

About the MESA Developer Studio Skin Editor

The MESA^{TM} Developer Studio Skin Editor provides an interface where you can quickly change the appearance or branding of Gentran Integration Suite. This includes basic page colors, images, and fonts. The Skin Editor allows you to edit all five of the main Gentran Integration Suite templates. You can change their properties so that they better reflect your company's brand image. The templates (pages) you can edit include:

- AFT/myAFT
- Dashboard
- Login
- MBI (mailbox)
- Admin
- Community Management

Basic properties include page colors, images, and fonts. Advanced properties include the use of Cascading Style Sheets (.css files). Changes are made through the Skin Editor Plug-in in the Eclipse interface. You download a copy of the current skin, make changes, then upload the revised skin to your Gentran Integration Suite installation. You can revert to the default skin at any time. Multiple skin versions can be saved for future use.

Note: Gentran Integration Suite must be running the first time MESA Developer Studio Skin Editor is used to verify the license and to download and deploy the skin.

Using the Skin Editor

When you open the Skin Editor, you must select a server from the Studio Tree View.

Retrieving Current Skin

Before editing a template, you must retrieve the current Gentran Integration Suite page skin from the instance. From the toolbar at the top of the page, select **Skin > Download skin**.

Note: If the current skin has not been downloaded, you are prompted to retrieve the skin from the instance.

Previewing a Skin

Once you have made changes to the template, you can view the changes by selecting Preview to open the selected page in the preview pane.

Note: When a template is edited, all other pages sharing the same skin are updated.

Saving Skin Changes

When the skin is ready to be deployed, select **Skin > Deploy changes.** The changes are saved to the remote Gentran Integration Suite instance.

Undoing Skin Changes

You can undo the last change made by selecting **Skin > Undo skin edit** or redo the last change made by selecting **Skin > Redo skin edit** from the toolbar at the top of the page.

The original skin can also be reapplied by selecting **Skin > Apply default skin**. The original skin will be restored to the remote instance.

Using the Skin Editor to Edit a Gentran Integration Suite Template

Once the existing Gentran Integration Suite skin is downloaded, you can use the Skin Editor to make changes to an existing template. The top left view provides a list of editable Gentran Integration Suite pages. Click on a page to open the property editor in the bottom left view, and open a preview window on the right.

To edit a template:

1. Open Eclipse and select the Skin Editor perspective.
2. Select the UI page type to be edited. The UI colors, image files, fonts, and properties appropriate for that page type are displayed.
3. Make desired changes and click **Save**. The changes are displayed in a preview.

   Note: You can refresh the view of the current skin at any time to see changes applied.
4. Specify the instance to apply the changes.
5. Test the connection to your instance.

6. Update the configured instance with the new look and feel information. The system confirms the update and displays a progress message.
7. You must restart/relaunch Gentran Integration Suite for the changes to take effect.

Using Advanced Editing

Use the advanced editing option in the Skin Editor to edit a Cascading Style Sheet (.css file) to your skin. If your company uses an existing style sheet for other applications, you can add it to the Skin Editor and use it to customize Gentran Integration Suite so that it will match your other applications.

You can also use advanced editing to views and makes changes to either the default UI settings, or the UI settings for the specified page type.

Restoring the Default Skin

You can restore the default skin that was included with Gentran Integration Suite at any time by selecting **Skin > Apply default skin**.

Gentran Integration Suite Architecture

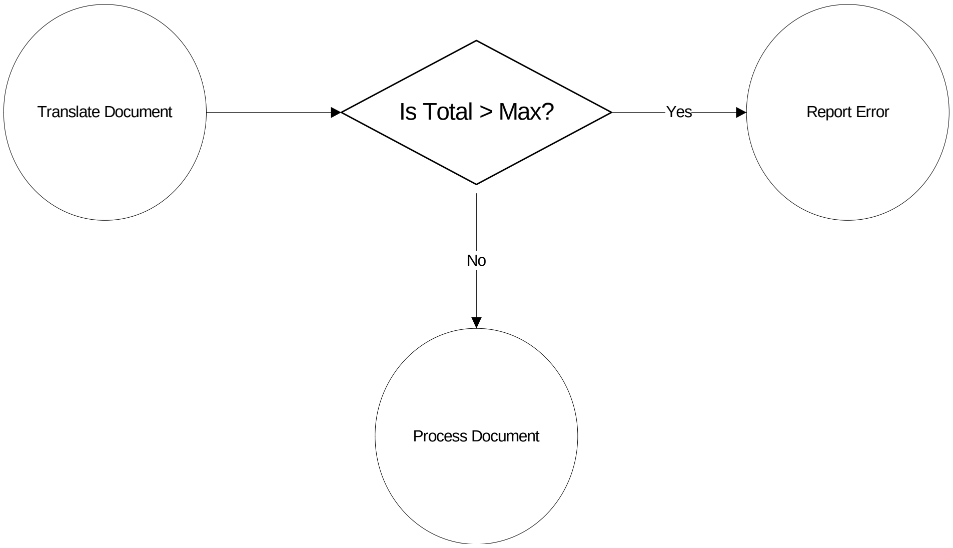
Introduction to Gentran Integration Suite Architecture

Gentran Integration Suite executes customer-specific business processes. An XML-based business process model directs the order of all processing activities in the application. This process model makes Gentran Integration Suite adaptable to a variety of processing situations.

Business Process Definitions

Gentran Integration Suite business process definitions are based on the draft Business Process Modeling Language (BPML) specification from the Business Process Management Initiative (www.bpmi.org). Because business process definitions are stored in XML, a business analyst can define business processes in several ways. An analyst can create definitions using a graphical editor, simple text editor, or any graphical process editor that can export the XML format recognized by Gentran Integration Suite.

The following figure shows a business process. A circle represents activities and a diamond represents a decision point.



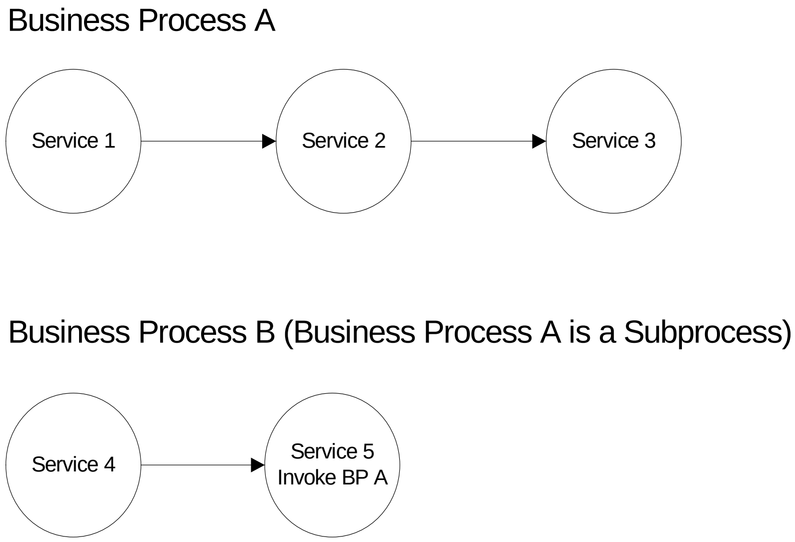
Services

Gentran Integration Suite views every activity in a business process as a *service*. A service can initiate:

- Legacy programs
- ERP systems
- Perl scripts
- Java code
- Decision engines
- Any computer program

A service in a business process can also invoke another business process, making that business process a subprocess. If the subprocess changes, the changes are reflected in all business processes that include that subprocess.

The following figure shows a subprocess:



Gentran Integration Suite supports reuse of business processes. Reuse enables you to determine what should be implemented as a service, a business process, or subprocess.

Reuse also enables a business analyst to work with information technology staff to determine whether the business processes should be written with multiple reusable components or as a single large service. For example, RosettaNet™ support can be implemented as multiple activities strung together to form a business process or as a single service.

Types of Services

There are several basic types of services in Gentran Integration Suite. The following table describes the various service types:

Type	Description
Internal	Services that are completely inside Gentran Integration Suite. Although internal services accept parameters and produce results, they do not directly interact with external systems (systems outside Gentran Integration Suite).
Input	Services that must receive data from external systems.
Output	Services that must send data to external systems.
Transport Adapter	Services that use communications protocols like FTP and HTTP to bring data into Gentran Integration Suite.
Application Adapter	Services that interact with external application systems.

Adapters

Adapters are services that interact with external systems.

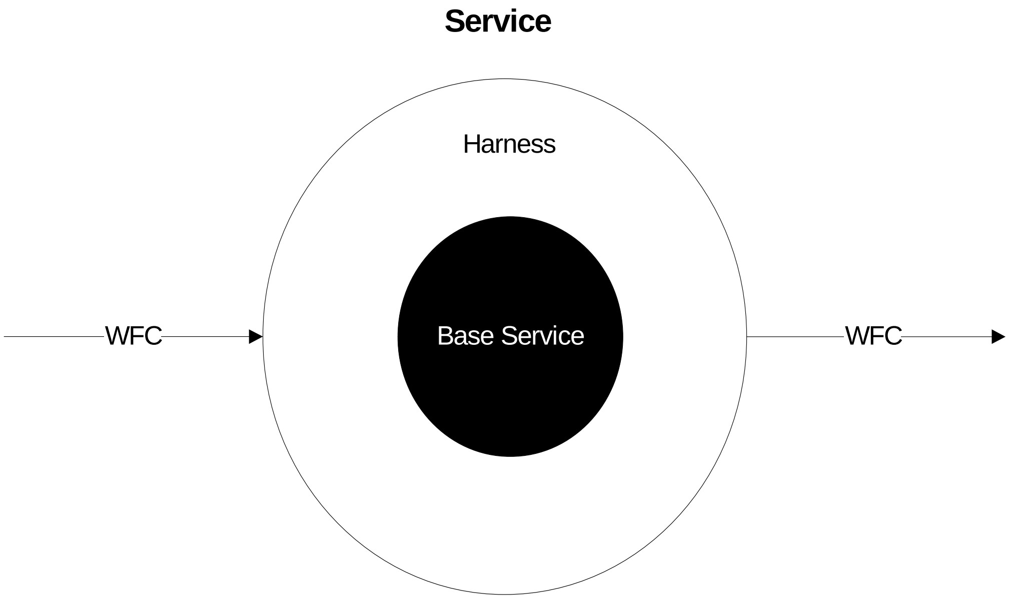
For more information about starting a business process, see *Starting a Business Process*.

Components of a Service

Every service accepts a business process state and produces a modified business process state or workflow context (WFC). Every service also has a harness. For the service, the harness performs the following functions:

1. Receives the input WFC
2. Extracts the information from it that the service needs
3. Runs the service
4. Places the results from the service in a new WFC or output for future steps in the business process workflow

The following figure shows a business process:



Example of a Service

To call an inventory system, the harness extracts the product ID from a document, sets up the calling parameters, sends the request to the external inventory system, extracts the inventory status from the return, and puts the appropriate status code in the workflow context.

Workflow Context

The *workflow context (WFC)* represents the business process state after each service has run. The WFC input to a service is written to a database. The service is complete after the new WFC is placed in persistent storage. Thus, the collection of persisted WFCs represents the state of all business processes in Gentran Integration Suite.

If Gentran Integration Suite stops, it can be restarted from the persisted WFCs by finding the most recent WFCs and sending those requests to the appropriate services. Internal services can be restarted automatically but external services require user intervention to restart them.

Basic Service Framework

The basic service framework or harness model enables Gentran Integration Suite to view all services similarly. For example, both the Translation service and the File System adapter have harnesses. Although these are different services, they support the same API as represented by the harnesses.

Sometimes an adapter harnesses a system that is outside the control of Gentran Integration Suite (for example, SAP®). Having a harness that presents a consistent interface to the rest of the system is very important. The harness is generic but the adapter is specific to the system with which it interacts to send and receive requests. Using the basic framework, you can start, configure, and stop an adapter for an external system in the Gentran Integration Suite interface. The actual operations of the external system are separate.

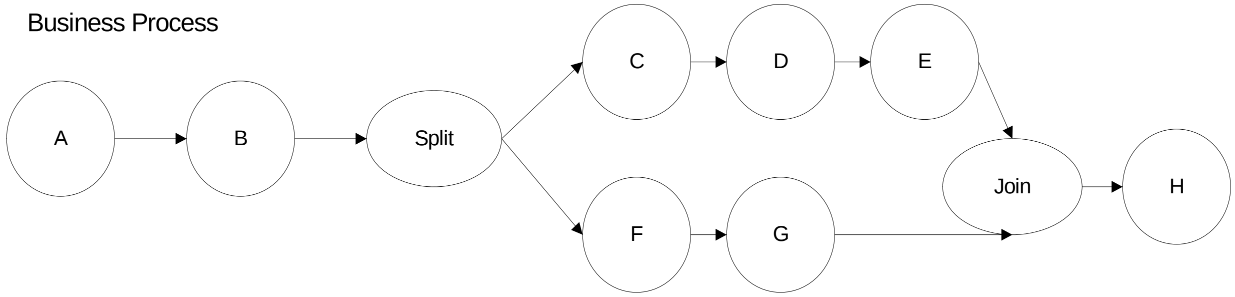
The harness also provides better performance. For example, the harness wrapped around the Translation service caches and reuses translation maps. The actual Translation service is unaffected by this action. This independence is especially important when the wrapped service is outside the control of Gentran Integration Suite.

Split and Join

Through the use of a split, a single business process can have multiple services running simultaneously.

A WFC reflects the status of only one thread or instance of the business process. A join enables the multiple instances to be collected to create a single WFC.

The following figure shows a business process with a split and a join:



Special Service Capabilities

Gentran Integration Suite supports the following unique capabilities, which afford you great flexibility with managing services:

- Large file support – The ability for services to handle files larger than available memory. This can be an effective way to help manage load sharing.
- Service groups – The ability to group "like" services together and treat them as a pool of services
- Storage types – The ability to select the document storage type for a service, such as Database, or File System

For more information about these topics, see the *Service and Adapter Guide*.

Relationship Between Business Processes and Services

Before you can run a business process definition, you must validate and compile it. Validation ensures that all activities in the business process definition are configured appropriately.

Compilation breaks the definition into smaller chunks: a header and entries. The header specifies global properties of the business process definition. Each service within the definition has an entry. The compiled information for each service includes all outbound edges, from that service to subsequent services and the parameters that they require. Outbound edges use service status to direct the flow of execution from one service to another.

Example of Outbound Edges

If Service A returns success or failure, its *success* outbound edge directs processing to continue to Service B, whereas its *failure* outbound edge directs processing to continue to Service X.

Gentran Integration Suite stores the compiled information for each service in the compiled ActivityInfo. ActivityInfo contains an abstract description of the service, such as *Translate the current document using map 5*.

Compilation enables Gentran Integration Suite to predetermine the start node of a business process. This capability makes the business process easy to instantiate and run and prevents Gentran Integration Suite from repeatedly parsing XML. Compilation also reduces the number of database queries because the next-step pointer is stored in the current activity information.

Starting a Business Process

Gentran Integration Suite supports dynamic selection (*bootstrapping*) of business processes. To specify dynamic selection of a business process, configure an adapter to select a business process definition by matching one or more adapter properties. For example, when a user creates a B2B HTTP Server adapter configuration, the user specifies a uniform resource identifier (URI) and then selects either a business process definition or contract specific to that URI. A user can create many B2B HTTP Server adapter configurations with different URIs, each of which invokes a business process or a contract.

Input data enters Gentran Integration Suite through an input adapter. An input adapter performs the following functions:

- Receives data from an external system
- Puts the data and any metadata into an initial workflow context (IWFC)
- Calls the IWFC start method to start the business process, which causes a business process definition to be found and instantiated for the input data

If the input data requires Gentran Integration Suite to start a new process, then the following steps take place:

1. A new WFC is created.
2. The WFC is put in persistent storage.
3. Gentran Integration Suite starts the associated business process.

Many-to-Many Relationship

Gentran Integration Suite bootstrapping creates a many-to-many relationship between adapters and business process definitions. Using the metadata given to the IWFC, one adapter can start several business processes.

Conversely, several adapters can start the same business process. A many-to-many relationship between adapters and business process definitions enables Gentran Integration Suite to focus on business problems, not just on how data arrives.

Making an input adapter the first step in a business process impairs the many-to-many relationship and keeps the business process from being reused as a subprocess.

Business Process Definition Fails to Start

If an adapter tries to start a business process definition that does not exist or is disabled, Gentran Integration Suite saves the request to start the business process definition and any related documents within Gentran Integration Suite. The user can use the business process monitor to view error messages for any business process definitions that failed to execute.

- ✦ If the business process definition cannot be found, the user can do an advanced restart and select a different business process definition, which uses the same input data.
- ✦ If the business process definition is disabled, then when the user enables that business process definition, Gentran Integration Suite automatically resumes any instances of that business process definition that stopped.

To ensure that an adapter catches the InitialWorkFlowContextException, code its logic accordingly:

```
{
    iwfc.start()
}
  catch (InitialWorkFlowContextException)
 {
 //do not delete our data here if this happens
 //set the appropriate response to the user
 }
```

To enable an adapter to start a business process with more than one document, code the following commands:

```
//for a single document
Document doc = new Document();
etc.
iwfc.putDocument(doc)
//for more than one document
Document doc1 = new Document();
etc.
Document doc2 = new Document();
etc.
iwfc.putDocument(name1, doc2);
iwfc.putDocument(name2, doc2);
```

name1 and *name2* are unique keys for the document within Gentran Integration Suite. When there is only one document, Gentran Integration Suite assigns the unique key of PrimaryDocument.

If a service needs to write more than one document, the service calls:

```
wfc.putDocument(name, doc);
```

For a single document, the service calls:

```
wfc.putDocument(doc))
```

To get a specific document from a set, the service calls:

```
wfc.getDocument(name)
```

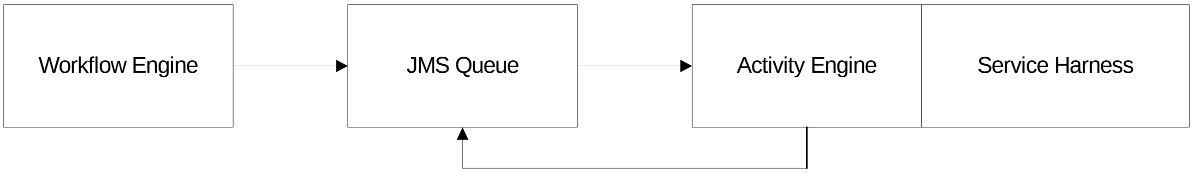
Running a Business Process

When a business process starts, the workflow engine (WFE) executes the services defined in the business process definition and the WFE creates a workflow context (WFC) from the initial workflow context (IWFC). The WFE uses the compiled ActivityInfo to get information about the first service to call. Next, the WFE puts the WFC on the JMS queue, so that the client initiating the business process does not wait for it to complete.

The WFE analyzes the compiled information to determine the current activity (service) that needs to be run. This information is stored in the compiled ActivityInfo. The ActivityInfo contains an abstract description of the service.

The WFE determines, for example, how the Translation service has been configured to run.

The following figure shows the execution cycle:



The JMS queue acts as a hand-off point. It does more than routing—it guarantees that the Java thread of execution is not interrupted during the running of the business process. The listener attached to the JMS queue is a lightweight activity engine. The activity engine takes the WFC off the JMS queue and invokes the service. Logically part of the workflow engine, the activity engine calls the service, takes the results from the service, and immediately starts the next cycle, determining the service that needs to be called and requesting that service on the JMS queue.

When configured properly, the usejms property of Gentran Integration Suite enables the activity engine to cycle-call the service directly. Cycle calling avoids the cost of using the JMS queue. However, cycle-calling also limits the clustering capabilities of application servers.

The activity engine can determine the next service because the harness has analyzed the service results and set the state values in the WFC. The activity engine consults these values to determine the next activity. The activity engine uses the return code from the current service to choose the next activity from a set of potential activities listed in the current ActivityInfo.

Gentran Integration Suite Components and a J2EE Environment

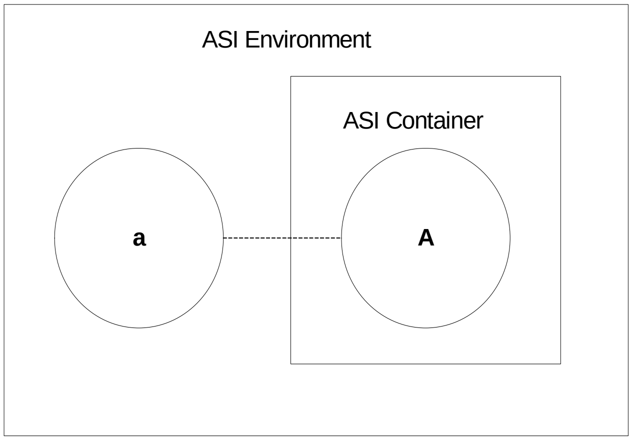
Gentran Integration Suite is written in Java and runs in a container proprietary to Sterling Commerce. This container allows Gentran Integration Suite to be independent of, yet integrate with popular J2EE application servers through standard means such as EJB and JCA clients. For the development of adapters, the service architecture provides a clear separation of concerns. There are two parts to each Adapter Implementation; the Service Harness Implementation that provides interaction and interface with the workflow engine, and the Service Adapter Implementation, which provides the interface to the external system. This allows the Service Adapter Implementation to run independently of the container if necessary, and gives more implementation options for the developer. For services that have no need to interface beyond the process boundary of Gentran Integration Suite, the implementation can be done with the harness.

Service Harness Implementation, Service Adapter Implementation

To overcome these restrictions, Gentran Integration Suite adapters are composed of two parts: *Service Harness Implementation*, the part of the adapter inside the ASI container; and *Service Adapter Implementation*, the part outside the ASI container. Services do not have a Service Adapter Implementation component.

The following figure shows an adapter implemented as Service Harness Implementation and Service Adapter Implementation:

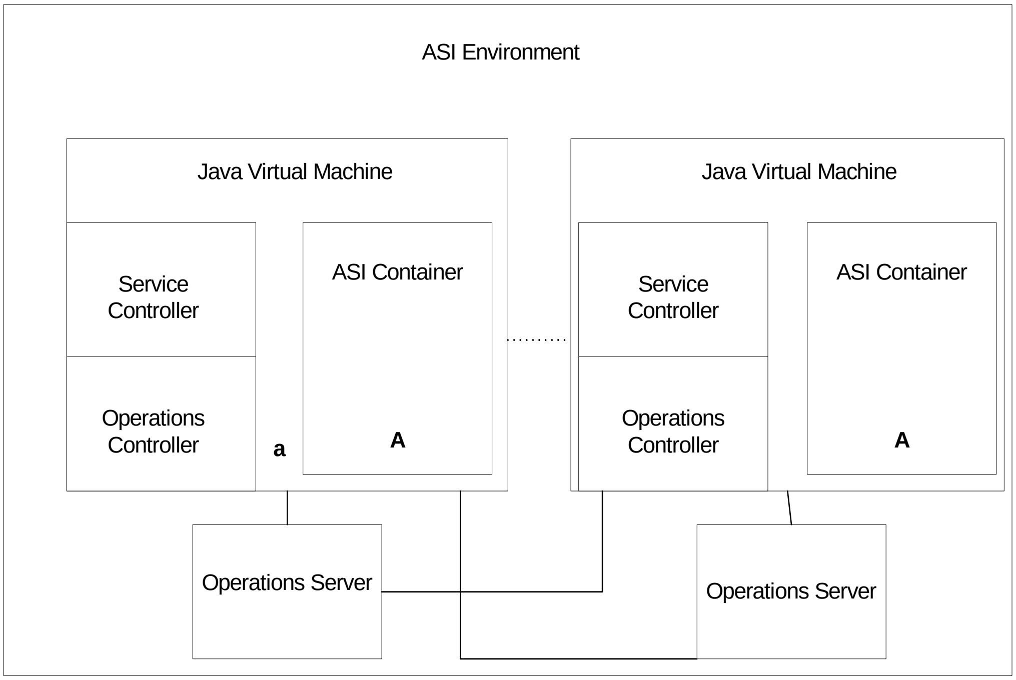
The Service Harness Implementation automatically scales and is portable across clusters because it is instantiated from within the ASI container. The Service Adapter Implementation is tied to a specific computer. The Service Harness Implementation can even move around from one call to the next. The Service Adapter Implementation, however, is fixed next to the resource it is accessing.



Example of Service Harness Implementation and Service Adapter Implementation

A cluster of computers in a ASI environment has private disk space on one computer. The Service Adapter Implementation portion of the File System adapter must be on the computer that can access the disk. The Service Harness Implementation portion of the File System adapter, however, can run in any container on any computer.

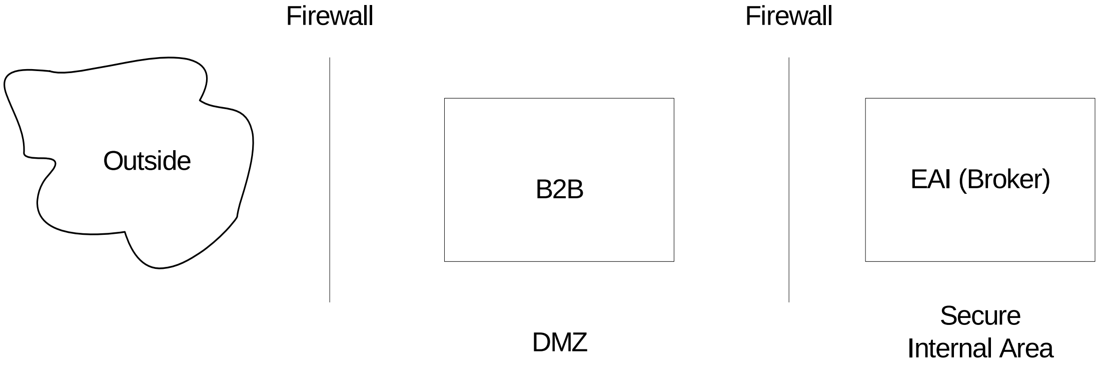
In the following figure, the Service Harness Implementation is moved to a different container in a different Java VM:



B2B Server

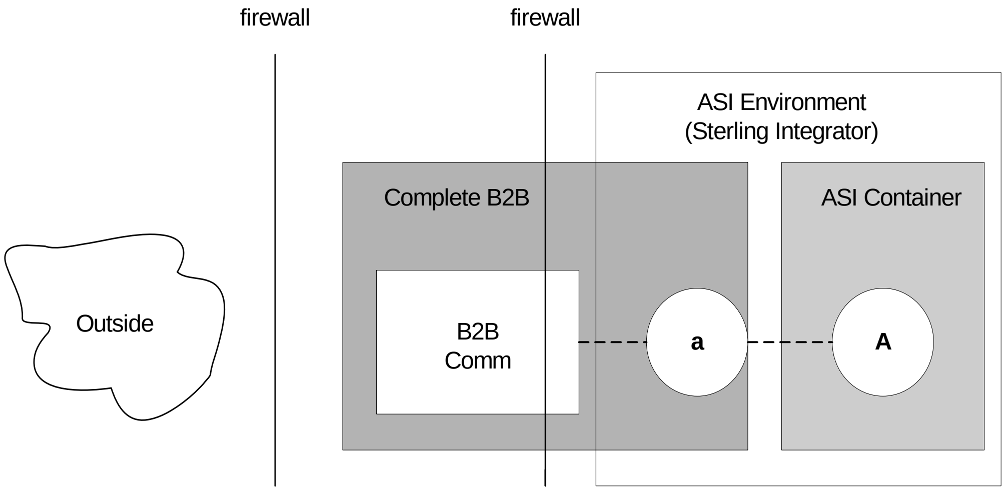
Gentran Integration Suite includes a business-to-business (B2B) server. The B2B server can be viewed as an independent system.

The following figure shows a traditional model of B2B and enterprise application integration (EAI):



However, within Gentran Integration Suite it is more appropriate to view the B2B server as a complex adapter. The B2B server has a two-part Service Adapter Implementation. One part runs in the DMZ and one part runs in the ASI environment.

The following figure shows a B2B server as a complex adapter:



Secure DMZ

The part of the B2B server that runs in the DMZ performs communications activities only. It stores no data. Trading profiles are stored in the secure area where Gentran Integration Suite resides. The part of the B2B server in the DMZ can run in a simple Java Virtual Machine (JVM) or a complete ASI environment.

The part of the B2B server in the DMZ and the part of the B2B server inside the secure area communicate as if they were separate systems and not part of a single ASI environment.

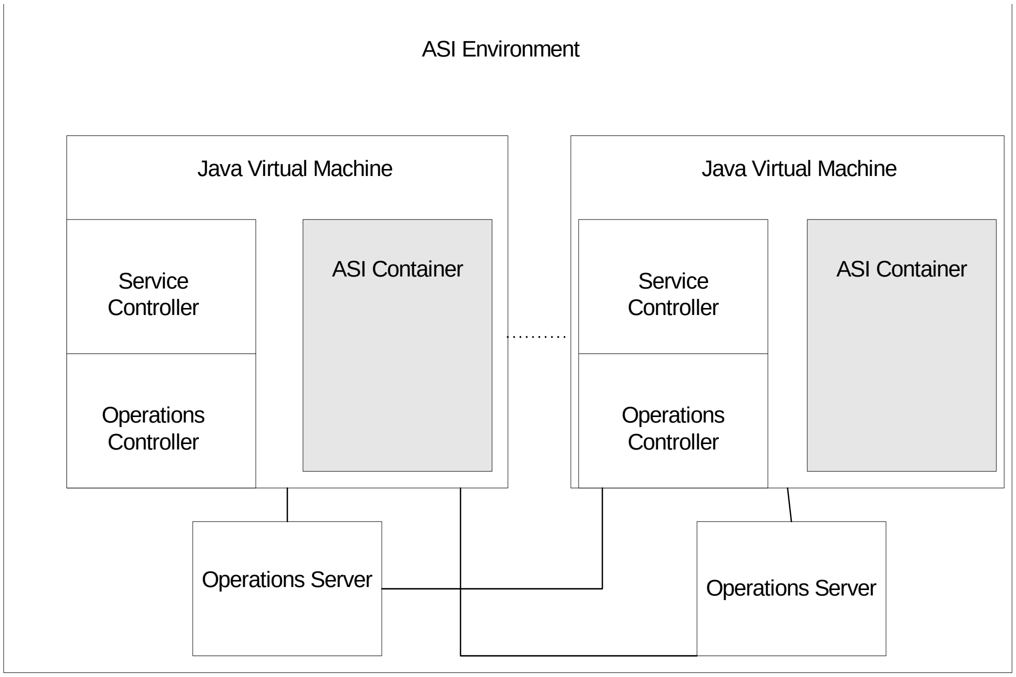
Service and Operations Controllers

The Gentran Integration Suite service and operations controllers monitor and manage executing services and workflows within the Gentran Integration Suite environment. These controllers free system operators and business analysts from having to attend to application-server details.

Service controllers provide a single place within a VM to manage, configure, query, and cache all service-related information. They also enable Gentran Integration Suite to scale and manage the Service Adapter Implementation parts of adapters. There is one service controller per VM in the ASI Container.

Operations controllers manage resources across VM boundaries. You can have multiple operations servers for redundancy and several embedded components, one per VM. Operations servers provide a single point of contact for all operational questions.

The following figure shows the service and operations controllers:



Reference Information for Developing a Service

In addition to knowing how to use MESA^{TM} Developer Studio SDK, you need to understand the following concepts to develop a service for Gentran Integration Suite.

This section covers the following topics:

- Service Architecture Summary
- Workflow Context
- Service Controller
- Error and Status Reporting
- Configuring Services
- Logging Service

Service Architecture Summary

An adapter interacts with external systems to get data in and out of Gentran Integration Suite. Typically, an adapter consists of a harness, a Remote Method Invocation (RMI), and files that enable the adapter to be used in the Gentran Integration Suite interface.

Harness

The harness part of the adapter must implement the processData() function, which the activity engine calls whenever it has work for the adapter to perform. This function can be called to push data out of the system or signal that data needs to be collected.

RMI (Service Adapter Implementation)

Preferably, most of an adapter workload should reside in the Service Harness Implementation. The Service Implementation exists mainly to allow for a separation of concerns from the harness.

Typically, adapters do some work that would be inefficient or hindering operating completely within the ASI Container. For example, the service implementation can be set up to wait for data to arrive and to periodically poll the external system for data.

Service XML File and Language Files

For a service to appear in the Gentran Integration Suite user interface and to be configured, XML entries must be added to the service.xml file with its associated language file.

For more information about language properties, see *Language-Specific Properties Files* on page 45 and *Service XML File* on page 45.

Scalability

Services in Gentran Integration Suite are scalable: For adapters, the Service Implementation creates a new thread to service each new request it receives. The Service Implementation is multi-threaded by default. You do not need to write additional code.

Workflow Context

The *workflow context API* encapsulates a basic unit of work, including all parameters required by the adapter to act on that unit of work.

The workflow context maintains the state of the business process from service to service. It contains, among other things, the document being manipulated by the business process. This is also where each service reports errors and status. The Gentran Integration Suite infrastructure is designed to persist the workflow context between steps.

The workflow context contains several components:

- *Input Parameters* – Retrieve parameters before beginning the operation
- *Workflow Document Body* – Set up the document body
- *Error Reporting* – Set up status and error reporting

Input Parameters

A service should have all of its parameters before doing any of its core logic. The workflow context provides the getWFContent method to retrieve parameters:

```
getWFContent(String parmName);
```

The getWFContent method retrieves a named input parameter from the workflow context. It gets global (service type), copy (service configuration), and WFD (workflow definition or business process definition) level parameters.

WFD parameters override service configuration parameters, which override service type parameters. If the workflow content message contains a string value with the parmName requested in the getWFContent method call, then the value in the workflow content hash table overrides all other values.

The hash table can contain any type of object. The getWFContent method enables a parameter to override a higher-level parameter only if its value is stored as a string in the hash table.

For example, the following string would retrieve an input parameter named *URL*:

```
String url = getWFContent("URL");
```

The service may need to get a parameter passed at run time by a previous service. If the parameter is not a service type, service configuration, or WFD parameter being overridden, then the service calls:

```
getWFContent(parmName);
```

Workflow Document Body

A typical service or adapter operates on a document contained within the workflow context. The document contains the body of the document, information about the name of the body, and metadata that describes the document.

To retrieve the document from the workflow context, use the getPrimaryDocument method:

```
getPrimaryDocument(document);
```

Here is an example:

```
Document doc = wfc.getPrimaryDocument();
if(doc == null) {  // no document?
   theAdapterLog.logError("Required document not found");
   wfc.setBasicStatusError();
   sci.unregisterThread();
   return wfc;
} // end if

byte[] body = doc.getBody();
```

After the document is retrieved, you can obtain the body of the document by using the getBody method:

```
getBody(body);
```

The following example shows a new document body inserted into the workflow context:

```
Document document = wfc.createDocument();
document.setBody(body);
document.setBodyName("somename");
wfc.putPrimaryDocument(document);
```

Error Reporting

Another important component of the workflow context is status and error reporting. The workflow engine requires an adapter to return status information at the completion of the requested activity. The requesting business process uses this information to make decisions that control business process flow. Status is reported within the workflow context.

For more information about error reporting, see *Error and Status Reporting* on page 43.

Service Controller

The *service controller* is a unified framework that all adapters use to remove application-server dependencies.

The service controller is also responsible for starting and stopping the adapter.

Stateless and Stateful Adapters

Stateless and stateful adapters differ at the object level. For stateless adapters, the service controller instantiates one object that services all configured copies of the adapter. Each request to the Service Implementation of the adapter must be a complete request, because states cannot be maintained between requests. For stateful adapters, the service controller instantiates one object for each configured copy of the adapter.

Instance variables for RMI objects are not useful because multiple threads (or, in the case of stateless adapters, multiple copies of the adapter) have access to the same instance variables, as if they were class variables.

Method variables are unique to the invocation of the method, so they are acceptable to use.

Service Controller Interface

An adapter is composed of two parts:

- A harness that is the interface to the workflow engine
- An RMI server that communicates with external systems

The following code example shows how the adapter processData method:

1. Registers with the service controller
2. Finds its RMI service
3. Invokes an RMI method
4. Unregisters with the service controller
5. Returns to the following code to the workflow engine:

```
String svcName = wfc.getServiceName;
ServicesControllerImpl sci = ServicesControllerImpl.getInstance();
sci.harnessRegister(new Integer(wfc.getWorkFlowId()).toString(),
   svcName);
        try{
            rmi = (Yourserver)sci.getAdapter(svcName);
        }
        catch(Exception e){
            wfc.setBasicStatusError();
            sci.unregisterThread();
            return wfc;
        }
        if ( rmi == null ){
            wfc.setBasicStatusError();
            sci.unregisterThread();
            return wfc;
        }
        try {
   // request “Service Adapter Implementation” to do work
         rmi.someRMIMethod(parms, xmlInBytes);
        }
        catch(Exception e) {
            wfc.setBasicStatusError();
            sci.unregisterThread();
            return wfc;
        }
```

This code example uses the following methods to accomplish its work:

Method	Description
harnessRegister(workflowID, serviceName)	Assists the service controller in its monitoring and control functions. The harnessRegister should be called at the beginning of the processData method. Returns – None.
UnregisterThread()	Assists the service controller in its monitoring and control functions. It should be called before the return of the processData method. Returns – None.

Service Controller Interface – RMI

The IAdapterImpl class provides the following methods for use in the Service Implementation of the adapter:

Method	Description
startup()	The service controller calls startup() after a stateful adapter is created. Startup() should perform all setup and initialization necessary for the adapter to function correctly. This method returns a Boolean value. A true return indicates that startup was successful.

Method	Description
shutdown()	The service controller calls the shutdown() method when a stateful adapter shutdown is required. shutdown() should perform all operations necessary to shut down the adapter. In the case of a multi-threaded adapter, shutdown() must ensure that all of its threads are stopped before returning to the service controller. shutdown() should wait for threads that are performing work directly for a workflow to become quiescent. The adapter can stop threads that are waiting for external input. The IAdapterImpl base class provides the methods interruptThreads() and stopThreads() to assist in shutting down errant threads. As an additional assistance to the shutdown() method, the base class provides a count of registered threads. This count can be found in the invokes variable. The shutdown() method should poll this variable no more than once a second, waiting for it to decrement to zero. Note that the shutdown thread does not appear in this count. If the adapter has threads that listen on sockets, the invokes count may never go to zero. The shutdown() method will need to account for this case. The service controller will not wait indefinitely for shutdown() to fulfill its responsibilities. The adapter can configure the time-out period by overriding getShutdownTimeout(). The service controller enables shutdown() the amount of time returned from getShutdownTimeout(), and then the shutdown thread is terminated. This method returns a Boolean value. A true return indicates that the shutdown() method completed successfully.
refresh()	The service controller calls refresh() when the configuration changes for a stateful adapter. refresh() should perform all setup and initialization necessary for the adapter to function correctly. If an adapter maintains a connection or connections to an end system, and the connectivity configuration changes, refresh() should return a false value to the service controller. In this case, the service controller shuts down the adapter and then restarts it. This action prevents workflows from trying to invoke the adapter while connectivity changes are occurring. This method returns a Boolean value. A true return indicates that refresh() completed successfully. A false return indicates that the adapter did not refresh, in which case the service controller shuts down the adapter and then restarts it.

Method	Description
getShutdownTimeout()	The service controller calls getShutdownTimeout() to determine how long to wait for shutdown to complete before terminating the adapter threads. A default implementation provided in the base class returns 60 seconds. This method returns the period for shutdown to complete, in milliseconds.
interruptThreads()	The interruptThreads method is provided in the base class to assist the adapter shutdown() method in shutting down its active threads. interruptThreads() calls the interrupt method on each active thread and assists in a graceful shutdown where possible. Returns – None.
stopThreads()	The stopThreads method is provided in the base class to assist the adapter shutdown() method for active threads. stopThreads() terminates every active thread. Returns – None.
registerThread()	The registerThread method assists the service controller in its monitoring and control functions. registerThread() should be called at the beginning of each method called by the Service Harness Implementation portion of the adapter. Returns – None.
unregisterThread()	The unregisterThread method assists the service controller in its monitoring and control functions. unregisterThread() should be called at the end of each method called by the Service Harness Implementation portion of the adapter. Returns – None.

Error and Status Reporting

The workflow engine requires a service to return status information at the completion of the requested activity. The requesting business process uses this information to make decisions that control business process flow.

Three types of status information are returned to the workflow engine:

- Basic status
- Advanced status
- Exceptions

Basic Status

Basic status is the overall status of the work performed by the service.

```
setBasicStatus(status)
```

For example:

```
setBasicStatus(com.sterlingcommerce.woodstock.workflow.WorkFlowContext.ERROR);
```

Advanced Status

The *advanced status* is a modifier for the basic status. Reporting the advanced status requires the service writer to analyze the service error categories. Business analysts use the list of advanced errors to test for error conditions. The list should be representative, but not long.

The workflow context provides the following method for setting advanced status:

```
setAdvancedStatus(String advancedStatus);
```

Exceptions

An *exception* indicates a possible failure condition. A service may need to generate a workflow exception when a required input parameter is:

- Missing
- Invalid
- Disabled. For example, a map or a workflow definition is disabled.

Use the following syntax to construct an exception:

```
new WorkFlowException(String errorText, int reasonCode)
```

Workflow exception reason codes are:

- public final static int GENERAL_PARM_ERROR = 0; (Default)
  Use this in situations that require, for example, missing properties files.
- public final static int MANDATORY_PARM_MISSING = 1;
- public final static int INVALID_VALUE_FOR_PARM = 2;
- public final static int RESOURCE_DISABLED = 3;
- public final static int NO_DOCUMENT = 4

The workflow engine places the error text string for the workflow exception into the status report.

Status Report

The service can add text describing the activity performed to the workflow context. The workflow context uses the following method for this purpose:

```
setWFStatusRpt(“Status_Report”, String statusReportText);
```

A status report is available to view if an Info icon appears in the Report column. Click the icon to display the status report for that service. You can view the status report text from the Gentran Integration Suite interface.

Successful Invocation

Use the Gentran Integration Suite interface to view the progress of a business process as it executes; use the business process monitor to view details after a business process has run. When a service is successfully invoked, the Status column displays *Success*. This display is a result of calling the setBasicStatusSuccess() method in the workflow context. In this case, the advanced status does not need to be set.

Unsuccessful Invocation

An unsuccessful invocation of an adapter or service should result in the Status column displaying *Error*. This display is a result of calling the setBasicStatusFailure() method in the workflow context. The setAdvancedStatus() method is also called to give additional information about the failure condition.

Configuring Services

The Gentran Integration Suite console can be set up to prompt for configuration information specific to the service being developed. Simple service configuration does not require you to write any Java code. The only requirement is that the service XML file is set up to specify the information to be collected.

To set up the Gentran Integration Suite console, you must use the following files:

- Language-specific properties files – Provide screen text in the language chosen by the user.
- Service XML file – Describes the information collected at configuration time.

Language-Specific Properties Files

The language-specific files should exist for each service XML file. It is also the custom to pick a two- or three-character service abbreviation for the service and prepend this abbreviation to each language-specific property name. For example, for a file system, *fs* would indicate file system and look like *fs.label* or *fs.description*. This convention helps to guarantee that the language-specific property names are unique.

Here is an example of a language-specific properties file:

```
fs.label = File System Adapter
fs.description = Collects and Extracts files from a file system.
fs.wfd.group1.title = Workflow Properties
fs.wfd.group1.instructions = Specify the appropriate workflow settings.
fs.instance.group1.title = Collection
fs.instance.group1.instructions = Specify the appropriate settings for collecting
data using the File System Adapter.
fs.action = Action
fs.cfolder = Collection Folder name
fs.efolder = Extraction Folder name
fs.pollinterval = Poll Interval (mins)
```

Service XML File

Here is an example of the service XML file:

```
SERVICE name="FileSystem" description="fs.description" label="fs.label"
implementationType="CLASS" JNDIName="FileSystemEJBHome" type="Adapter"
adapterType="STATELESS"
adapterClass="com.sterlingcommerce.woodstock.services.filesystem.FileSystemServerImp
l" version="1.0" SystemService="NO">
```

The following table describes the variables in the service XML file:

Variable	Description
name	Descriptive name of the service.
description	Description in language-specific form.
label	Descriptive name in language-specific form.
implementationType	Either RMI or CLASS.
JNDIName	Lookup name of the service.
type	Either Adapter, basic, Advanced, Split, or Join.
adapterType	Either STATEFUL or STATELESS.
adapterClass	Class name of the Service Implementation of the adapter.
version	1.0
systemService	NO for adapters.

<VARS> Tags

Inside the <Service> tag are <VARS> tags. The <VARS> tags contain definitions of configuration items to collect from the user. Three types of <VARS> tags correspond to the scope of the configuration:

- global – The widest possible scope, applicable to all adapters of this type. Configuration parameters are displayed in the **Deployment > Services > Installation/Setup** section.
- instance – Limited in scope to a single instance of an adapter. Configuration parameters are displayed in the **Deployment > Services > Configuration** section.
- wfd – Workflow definition configuration for use only by the Graphical Process Modeler. The primary purpose of this tag type is to define the possible configuration that can be made in the workflow definition. Because the configuration is defined here, the Graphical Process Modeler can display the possible configuration to the user.

<VARS> Tag Example

Following is a sample <VARS> tag:

```
<VARS type="instance">
```

<GROUP> Tags

Inside the <VARS> tags are <GROUP> tags. <GROUP> tags group configuration information by page. The <GROUP> tag is part of the <VARS> tag and contains title and instructions.

- title – The title of the current page

✦ instructions – Help about what the user is supposed to do with the current page.

<Group> Tags Example

Following is a sample <Group> tag:

```
<GROUP title="fs.instance.group1.title"
instructions="fs.instance.group1.instructions">
```

<VARDEF> Tag

Inside the <GROUP> tag is the <VARDEF> tag.

The following table describes the <VARDEF> tag elements:

Tag Type	Description
varname	Name of the Java property that a service uses to retrieve data collected from the user.
type	Type of the input, normally String.
htmlType	Type of input you are retrieving from the user. Normal textual information is htmlType *text*, radio buttons are specified with *radio*, and password information can be retrieved with the htmlType *Password*. For text area, specify *textarea*. For drop-down list, specify the htmlType *select*.
validator	Validation types—for example, ALPHANUMERIC specifies that only alphabetic and numeric characters are accepted as input. NUMBER and NUMERIC specify only numeric validation.
label	Description of the configuration to be entered by the user.
size	Field size that displays to the user when collecting information.
maxsize	Maximum number of characters the user can enter for this variable.
required=NO	Specified only if the variable is not required.
defaultVal	Default value that is useful only if the VARS type is wfd.

The following example shows how all of the tags work together:

```
<VARS type="instance">
    <GROUP title="fs.instance.group1.title"
instructions="fs.instance.group1.instructions">
<VARDEF varname="collectionFolder" type="String" htmlType="text"
validator="ALPHANUMERIC" size="30" maxsize="250" label="fs.cfolder" />
     <VARDEF varname="useSubFolder" type="String" htmlType="radio"
validator="ALPHANUMERIC" options="radio2" label="fs.subfolders" />
    </GROUP>
</VARS>
```

Logging Service

This section provides general guidelines for logging from a service in the Gentran Integration Suite framework. The *Logging service* is a unified framework for logging messages.

Because services may have different needs and functions, apply the following guidelines only with a good understanding of the service logging output.

Logging Event Guidelines

The following table lists the guidelines for the types of events to log for each logging method:

Method	Guideline
Error	Log any non-exception occurrence that would cause the intended operation of the service to fail. Examples ◆ Communication failed to external system ◆ Invalid input data
Exception	Log any checked exceptions that are the result of an error or fault. Exceptions that are handled by the service should not be logged with logException. Example Any caught exception that is not handled by the adapter
Warn	Log any system disruption that is neither fatal nor a handled exception. Also log any notifications of possible error conditions. Examples ◆ Connection to external resource is lost while no processing is occurring ◆ Disk space limits ◆ License file expirations
Debug	Log anything that is useful information for development/debugging purposes, including general processing information and functionally relevant occurrences. Examples ◆ Entering and exiting of major methods ◆ Method input parameters ◆ Method return values ◆ Parameter values (setting and getting) ◆ Initiation and completion of operations

Method	Guideline
Log	Log anything that should always appear in the log file. This method sends messages to the default log regardless of log level. Examples ◆ Adapter StartUp ◆ Adapter ShutDown ◆ Adapter Refresh ◆ Connection to external systems

XLogger Logging

The XLogger class provides a unified log output, which includes the following:

- Module name
- Thread ID
- Service name

This class can also be used in stateful RMI adapters (Service Adapter Implementation's).

For more information about RMI logging, see *RMI Logging* on page 49.

EJB Logging

In the Service Harness Implementation of the adapter, the service name is available in the workflow context and can be used in the constructor when creating an XLogger copy.

For example, use this form of the constructor at the beginning of the processData method:

```
adapterLogger = new XLogger(classsName, getServiceName());
```

The adapterLogger should be an instance variable of type XLogger.

RMI Logging

For the Service Implementation of the adapter there are two approaches, depending on whether the adapter is stateless or stateful.

For stateless adapters, there is only one object created for all copies of the adapter. Therefore, instance variables are shared by all copies of the adapter.

The service name is not available to the Service Implementation of a stateless adapter from the Gentran Integration Suite infrastructure. The EJB must pass the service name to the Service Implementation of the adapter. The service name should not be used in the constructor when creating an XLogger copy—for example, adapterLogger = new XLogger(classsName). AdapterLogger should be a class variable of type XLogger. In this case, use the logging method that includes the service name.

For stateful adapters, there is an object created for each copy of the adapter, but all the threads of that adapter copy share the same object. Therefore, all threads of an adapter copy share instance variables.

The service name is available to the Service Implementation of a stateful adapter from the Gentran Integration Suite infrastructure by calling getServiceName().

The following is sample code:

```
adapterLogger = new XLogger(classsName, getServiceName());
```

The adapterLogger should be a class variable of type XLogger. The easiest way to do this is to construct a new XLogger if one does not already exist.

XLogger Logging Methods

You can use the XLogger logging methods in the EJB and RMI parts of the adapter. However, the availability of the service name varies.

Use the XLogger methods for logging from adapters. These methods format the class name and the service name provided along with a thread identifier, and add the message to be logged. The result is a log message that includes date/time, class name, service name, thread ID, and message.

The following table describes the XLogger class methods and indicates where they are used:

Method	**Used by Stateful RMI and EJBs**	**Used by Stateless RMI**
XLogger(String ClassName)		Constructor
XLogger(String ClassName, String ServiceName)	Constructor	
logError(String ServiceName, String message)		Logging errors
logError(String message)	Logging errors	
logException(String ServiceName, String message, Exception e)		Logging fault exceptions
logException(String message, Exception e)	Logging fault exceptions	
logWarn(String ServiceName, String message)		Logging warning messages
logWarn(String message)	Logging warning messages	
logDebug(String ServiceName, String message)		Logging debug messages
logDebug(String message)	Logging debug messages	
log(String ServiceName, String message)		General logging
log(String message)	General logging	

LogService Logging Methods

The *LogService class* provides the following static logging methods. Use these methods at any time in an EJB or RMI part of an adapter. Because each method implies a different logging threshold, some general usage guidelines are provided.

Each log message generated through LogService has a timestamp and logging level. You must supply the name of the originating class and a message.

The following sample code shows an example:

```
[2001-06-12 12:46:55.381] ALL [SiebelEJBBean] Hello World
```

Timestamp	Logging Level	Originating Class	Message
`[2001-06-12 12:46:55.381]`	`ALL`	`[SiebelEJBBean]`	`Hello World`

The following table lists the log methods and general guidelines for their formatting:

Note: For the BPML specifications that Gentran Integration Suite accepts, see the Gentran Integration Suite *Business Process Guide*.

Method	Format Guideline	Example
logError()	[Class name] General business error. Specific application error.	[SiebelEJBBean] Error: Adapter unable to process request. Action parameter is null.
logException()	[Class name] General business error. Specific application error. The logException() method requires both a message and an exception. This method logs the exception name and stack trace.	[SiebelEJBBean] Exception: Adapter unable to process Read request. Unable to connect to the file system server.
logWarn()	[Class name] General Warning. Expected results or recommended actions to be taken.	[SiebelEJBBean] Warning: Available disk space is less than 1 GB. Delete unnecessary files.
logDebug()	[Class name] Debug message	[SiebelEJBBean] Entering ProcessData method.
log()	[Class name] Log message	[SiebelEJBBean] Adapter started.

File System Adapter Examples

This appendix contains code examples of a working, fully functional File System adapter. Use these examples as a reference to help you to create your own adapter.

This appendix contains the following sample files:

- FileSystemServer File
- FileSystemServerImpl File
- File System Adapter XML
- FileSystem XML
- Filesystem_en File
- FileSystemImpl File
- FileSystemServer File
- FileSystemServerImpl File
- FSFromCollectInfo File
- FSFromReadFile File
- FSToCollectInfo File
- FSToExtractInfo File
- WFStartThread File

FileSystemServer File

The following sample code shows a FileSystemServer.java file:

```
/**
 * Copyright: Sterling Commerce, 2000-2001. All rights reserved.
 *
 * This software is the proprietary information of
   Sterling Commerce,
 * Inc. Use is subject to license terms.
 */
package com.sterlingcommerce.woodstock.services.filesystem;

import weblogic.rmi.Remote;
import weblogic.rmi.RemoteException;
import com.sterlingcommerce.woodstock.services.AdapterException;
import com.sterlingcommerce.woodstock.services.IAdapterRMI;

/**
 * Title:       FileSystem Adapter project
 * Description: This adapter can collect or extract files
     to a file system
 * Copyright:   Sterling Commerce, 2000. All rights reserved.
 * @version 1.0, 2/6/2001
 * @since Woodstock 2.0
 */
public interface FileSystemServer extends IAdapterRMI
{
    String[] scanFolder(String folder, String fileFilter, boolean
      useSubFolders) throws AdapterException, RemoteException;

    void createFile(String absoluteFileName) throws
      AdapterException, RemoteException;

    byte[] readFile(String absoluteFileName) throws
      AdapterException, RemoteException;

    void writeFile(String absoluteFileName, byte[] buffer) throws
      AdapterException, RemoteException;

    void deleteFile(String absoluteFileName) throws
      AdapterException, RemoteException;
}
```

FileSystemServerImpl File

The following sample code shows a FileSystemServerImpl.java file:

```
/**
 * Copyright: Sterling Commerce, 2000-2001. All rights reserved.
 *
 * This software is the proprietary information of
   Sterling Commerce,
 * Inc. Use is subject to license terms.
 */
package com.sterlingcommerce.woodstock.services.filesystem;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.BufferedInputStream;
import java.io.BufferedOutputStream;
import java.util.Vector;
import java.util.*;
import java.util.Properties;
import com.sterlingcommerce.woodstock.util.WildCardFilter;
import com.sterlingcommerce.woodstock.util.frame.Manager;
import com.sterlingcommerce.woodstock.services.*;
import com.sterlingcommerce.woodstock.workflow.*;
import com.sterlingcommerce.woodstock.util.frame.log.*;

/**
 * Implements the I/O routines for the File System EJB
 * @version 1.0, 2/6/2001
 * @since Woodstock 2.0
 */
public class FileSystemServerImpl extends IAdapterImpl implements FileSystemServer
{
    public FileSystemServerImpl() { super(); }
    public void refreshAdapter(Properties p) {}
    public String message(String s) { return s; }
    public void startupAdapter(Properties p) {}
    public void shutdownAdapter() {}

    /**
     * Method for returning an array of filenames in a directory
     * @param folderName name of the directory
     * @param useSubFolders
     * @since Woodstock 2.0
     */
    public String[] scanFolder(String folderName, String fileFilter, boolean
useSubFolders)
    {
        registerThread();
        Vector fileVect = new Vector();
        WildCardFilter filter = new WildCardFilter(fileFilter);
        traverseDir(fileVect, filter, folderName, useSubFolders);
        String[] fileNames = null;
        int vectSize = fileVect.size();
        if ( vectSize > 0 )
        {
            fileNames = new String[vectSize];
            while ( vectSize-- > 0 )
            {
                fileNames[vectSize] = (String)fileVect.elementAt(vectSize);
            }
        }
        unregisterThread();
        return fileNames;
    }

    /**
     * Method for recursively traversing a directory structure
     * @param fileVect Vector object used to collect the recursed information
     * @param folderName name of the directory
     * @param useSubFolders
     * @since Woodstock 2.0
     */
    public void traverseDir(Vector fileVect, WildCardFilter filter, String
folderName, boolean useSubFolders)
    {
        if ( folderName != null )
        {
            File folder = new File(folderName);
            File[] fileList = folder.listFiles(filter);
            if ( fileList != null )
            {
                for ( int i = 0; i < fileList.length; i++ )
                {
                    if ( !fileList[i].isDirectory() )
                    {
                        fileVect.add(fileList[i].getAbsolutePath());
                    }
                    else if ( useSubFolders )
                    {
                        traverseDir(fileVect, filter, fileList[i].getAbsolutePath(),
useSubFolders);
                    }
                }
            }
        }
    }

    /**
     * Method for creating a file on disk
     * @param absoluteFileName name of the file with its absolute path
     * @exception AdapterException
     * @since Woodstock 2.0
     */
    public void createFile(String absoluteFileName) throws AdapterException
    {
        registerThread();
        try
        {
            new File(absoluteFileName);
        }
        catch(Exception e)
        {
            unregisterThread();
            throw new AdapterException(e);
        }
        unregisterThread();
    }

    /**
     * Method for deleting a file on disk
     * @param absoluteFileName name of the file with its absolute path
     * @exception AdapterException
     * @since Woodstock 2.0
     */
     public void deleteFile(String absoluteFileName) throws AdapterException
     {
        registerThread();
        File file = null;
        try
        {
            file = new File(absoluteFileName);
            file.delete();
        }
        catch(Exception e)
        {
            unregisterThread();
            throw new AdapterException(e);
        }
        unregisterThread();
     }

    /**
     * Method for reading a file from disk.<p>
     * @param absoluteFileName name of the file (inc. its absolute path name)
     * @return byte[] array of bytes making up the data
     * @exception AdapterException
     * @since Woodstock 2.0
     */
    public byte[] readFile(String absoluteFileName) throws AdapterException
    {
        registerThread();
        BufferedInputStream biStream = null;
        File file = new File(absoluteFileName);
        long fileSize = 0;
        int cbRead = 0;
        byte[] buffer = null;

        try
        {
            biStream = new BufferedInputStream(new FileInputStream(file));
            fileSize = file.length();
            buffer = new byte[(int)fileSize];
            cbRead = biStream.read(buffer, 0, (int)fileSize);
        }
        catch(Exception e)
        {
            unregisterThread();
            throw new AdapterException(e);
        }
        finally
        {
            if ( biStream != null )
            {
                try
                {
                    biStream.close();
                }
                catch(Exception e)
                {
                    LogService.out.logWarn(e.getMessage());
                }
            }
        }

        unregisterThread();
        if ( cbRead > 0 )
        {
            return buffer;
        }
        else
        {
            return null;
        }
    }

    //JOE - STUFF
    private static Hashtable threadTable = new Hashtable();
    private final static int thresh = 10; // only do 10 threads

    public void workflowStart(String wfId, String name, byte[] stuff, String path,
String svcName) throws AdapterException
    {
        synchronized(threadTable)
        {
            registerThread();
            try
            {
                int iwfId = -1;
                if ( wfId != null && !wfId.equals("") )
                {
                    try
                    {
                        iwfId = Integer.parseInt(wfId);
                    }
                    catch(NumberFormatException nfe)
                    {
                        iwfId = -1;
                    }
                }
                InitialWorkFlowContext iwfc = new InitialWorkFlowContext();
                iwfc.setWorkFlowDefId(iwfId);
                iwfc.setDocumentName(name);
                iwfc.setDocumentBody(stuff);

                if ( threadTable.size() > thresh )
                {
                    waitForComplete();
                }

                WFStartThread wfst = new WFStartThread(iwfc, path, svcName);
                Thread t = new Thread(wfst);
                String id = "FS." + wfId + "." + name + "." +
System.currentTimeMillis() + ":" + iwfc.hashCode();
                threadTable.put(id, t);
                t.start();
            }
            catch(Exception e)
            {
                unregisterThread();
                throw new AdapterException(e);
            }
            unregisterThread();
        }
    }
    private void waitForComplete()
    {
        Thread t = null;
        Enumeration e = null;
        boolean again = true;
        String k;
        int ct = 0;
        while ( again && ct < 100000 )
        {
            e = threadTable.keys();
            while ( e.hasMoreElements() )
            {
                k = (String)e.nextElement();
                t = (Thread)threadTable.get(k);
                if ( !t.isAlive() )
                {
                    threadTable.remove(k);
                    again = false;
                }
            }
            ct++;
            try
            {
                Thread.sleep(100);
            }
            catch(Exception ex)
            {
                ex.printStackTrace();
            }
        }
    }

    /**
     * Method for writing a file to disk.<p>
     * @param absoluteFileName name of the file with its absolute path
     * @param buffer array of byte to write to the file
     * @exception AdapterException
     * @since Woodstock 2.0
     */
    public void writeFile(String absoluteFileName, byte[] buffer) throws
AdapterException
    {
        registerThread();
        BufferedOutputStream boStream = null;
        try
        {
            boStream = new BufferedOutputStream(new FileOutputStream(new
File(absoluteFileName)));
            boStream.write(buffer, 0, buffer.length);
        }
        catch(Exception e)
        {
            unregisterThread();
            throw new AdapterException(e);
        }
        finally
        {
            if ( boStream != null )
            {
                try
                {
                    boStream.flush();
                    boStream.close();
                }
                catch(Exception e)
                {
                    LogService.out.logWarn(e.getMessage());
                }
            }
        }
        unregisterThread();
    }
}
```

File System Adapter XML

The following sample code shows a File System adapter XML:

```
<SERVICES>

<SERVICE name="FileSystem" description="fs.description" label="fs.label" 
implementationType="EJB" JNDIName="FileSystemEJBHome" type="Adapter" 
adapterType="STATELESS" 
adapterClass="com.sterlingcommerce.woodstock.services.filesystem.FileSystemServerImp
l" version="1.0" SystemService="NO">
 <BP_XML>
    <![CDATA[<process name="Scheduler_&service_name;">
       <sequence>
          <operation name="Service">
             <participant name="&service_name;"/>
             <output message="Xout">
                <assign to="." from="*"></assign>
                <assign to="Action">FS_COLLECT</assign>
             </output>
             <input message="Xin">
                <assign to="." from="*"></assign>
             </input>
          </operation>
       </sequence>
    </process>]]>
 </BP_XML>
 <VARS type="wfd">
    <GROUP title="fs.wfd.group1.title" instructions="fs.wfd.group1.instructions">
       <VARDEF varname="Action" type="String" htmlType="select" 
validator="ALPHANUMERIC" label="fs.action" options="fstype" />
    </GROUP>
  </VARS>
  <VARS type="instance">
```

```
    <GROUP title="fs.instance.group1.title"
instructions="fs.instance.group1.instructions">
      <VARDEF varname="collectionFolder" type="String" htmlType="text"
validator="ALPHANUMERIC" size="30" maxsize="250" label="fs.cfolder" />
     <VARDEF varname="useSubFolder" type="String" htmlType="radio"
validator="ALPHANUMERIC" options="radio2" label="fs.subfolders" />
    </GROUP>
    <GROUP title="fs.instance.group2.title"
instructions="fs.instance.group2.instructions">
      <VARDEF varname="extractionFolder" type="String" htmlType="text"
validator="ALPHANUMERIC" size="30" maxsize="250" label="fs.efolder" />
      <VARDEF varname="assignFilename" type="String" htmlType="radio"
validator="ALPHANUMERIC" options="fsfilename" label="fs.filename">
         <SUBGROUP dependencyvalue="true" title="fs.instance.group2a.title"
instructions="fs.instance.group2a.instructions">
            <VARDEF varname="assignedFilename" type="String" htmlType="text"
validator="ALPHANUMERIC" size="40" maxsize="250" label="fs.extractfilename" />
         </SUBGROUP>
      </VARDEF>
    </GROUP>
    <GROUP title="system.sched.title" instructions="system.sched.instructions">
     <VARDEF varname="schedDay" type="String" htmlType="select" validator="NUMBER"
label="system.schedDay.label" options="schedDay"/>
     <VARDEF varname="schedHour" type="String" htmlType="select" validator="NUMBER"
label="system.schedHour.label" options="schedHour"/>
    <VARDEF varname="schedMinute" type="String" htmlType="select" validator="NUMBER"
label="system.schedMinute.label" options="schedMinute"/>
     <VARDEF varname="schedOnMin" type="String" htmlType="select"
validator="ALPHANUMERIC" label="system.schedOnMin.label" options="schedOnMinute"/>
     <VARDEF varname="schedMerid" type="String" htmlType="select" validator="NUMBER"
label="system.schedMerid.label" options="schedMeridian"/>
    </GROUP>
  </VARS>
  <VARS type="assignbp">
    <GROUP title="bpsched.assignbp.title"
instructions="bpsched.assignbp.instructions">
      <VARDEF varname="initialWorkFlowId" type="String" htmlType="select"
validator="ALPHANUMERIC" label="bpsched.assignbp.title" options="bplist" />
    </GROUP>
  </VARS>
</SERVICE>

<OPTION name="fstype">
    <ELE value="FS_COLLECT" displayname="fs.Collection" />
    <ELE value="FS_EXTRACT" displayname="fs.Extraction" />
</OPTION>

<OPTION name="fsfilename">
    <ELE value="false" displayname="fs.useOriginal" />
    <ELE value="true" displayname="fs.assignName" />
</OPTION>

</SERVICES>
```

FileSystem XML

The following sample code is a FileSystem XML file:

```
-->
- <SERVICES>
- <SERVICE name="FileSystem" description="fs.description" label="fs.label"
implementationType="CLASS"
JNDIName="com.sterlingcommerce.woodstock.services.filesystem.FileSystemImpl"
type="Adapter" adapterType="STATELESS"
adapterClass="com.sterlingcommerce.woodstock.services.filesystem.FileSystemServerImp
l" version="1.0" SystemService="NO">
- <BP_XML>
- <![CDATA[
<process name="Scheduler_&service_name;">
       <sequence>
          <operation name="Service">
             <participant name="&service_name;"/>
             <output message="Xout">
                <assign to="." from="*"></assign>
                <assign to="Action">FS_COLLECT</assign>
             </output>
             <input message="Xin">
                <assign to="." from="*"></assign>
             </input>
          </operation>
       </sequence>
    </process>
]]>
</BP_XML>
- <VARS type="wfd">
- <GROUP title="fs.wfd.group1.title" instructions="fs.wfd.group1.instructions">
<VARDEF varname="Action" type="String" htmlType="select" validator="ALPHANUMERIC"
label="fs.action" options="fstype" />
<VARDEF varname="appendOnExtract" type="String" htmlType="select"
validator="ALPHANUMERIC" label="fs.append" options="radio2" />
<VARDEF varname="deleteAfterCollect" type="String" htmlType="select"
validator="ALPHANUMERIC" label="fs.delete" options="radio2" />
<VARDEF varname="collectZeroByteFiles" type="String" htmlType="select"
validator="ALPHANUMERIC" label="fs.zero" options="radio2" />
<VARDEF varname="fileModTimeThreshold" type="String" htmlType="text"
validator="ALPHANUMERIC" size="30" maxsize="250" label="fs.modtime" />
<VARDEF varname="maxThreads" type="String" htmlType="text" validator="NUMBER"
size="5" maxsize="5" label="fs.maxthreads" />
</GROUP>
</VARS>
- <VARS type="instance">
- <GROUP title="fs.instance.group1.title"
instructions="fs.instance.group1.instructions">
<VARDEF varname="collectionFolder" type="String" htmlType="text"
validator="ALPHANUMERIC" size="30" maxsize="250" label="fs.cfolder" />
<VARDEF varname="filter" type="String" htmlType="text" size="30" maxsize="250"
label="fs.filter" required="NO" />
```

```
<VARDEF varname="useSubFolders" type="String" htmlType="radio"
validator="ALPHANUMERIC" options="radio2" label="fs.subfolders" />
<VARDEF varname="keepPath" type="String" htmlType="radio" validator="ALPHANUMERIC"
options="radio2" label="fs.keepPath" defaultVal="false" />
- <VARDEF varname="bootstrap" type="String" htmlType="radio" validator="ALPHANUMERIC"
options="radio2" label="fs.bootstrap">
- <SUBGROUP dependencyvar="bootstrap" dependencyvalue="true"
title="bpsched.assignbp.title" instructions="bpsched.assignbp.instructions">
<VARDEF varname="initialWorkFlowId" type="String" htmlType="select"
validator="ALPHANUMERIC" label="bpsched.assignbp.title" options="bplist" />
</SUBGROUP>
<SUBGROUP dependencyvar="bootstrap" dependencyvalue="true"
handler="com.sterlingcommerce.woodstock.ui.ScheduleConfig" wizard="Scheduler" />
</VARDEF>
</GROUP>
- <GROUP title="fs.instance.group2.title"
instructions="fs.instance.group2.instructions">
<VARDEF varname="extractionFolder" type="String" htmlType="text"
validator="ALPHANUMERIC" size="30" maxsize="250" label="fs.efolder" />
- <VARDEF varname="assignFilename" type="String" htmlType="radio"
validator="ALPHANUMERIC" options="fsfilename" label="fs.filename">
- <SUBGROUP dependencyvalue="true" title="fs.instance.group2a.title"
instructions="fs.instance.group2a.instructions">
<VARDEF varname="assignedFilename" type="String" htmlType="text" size="40"
maxsize="250" label="fs.extractfilename" />
</SUBGROUP>
</VARDEF>
</GROUP>
</VARS>
</SERVICE>
- <OPTION name="fstype">
<ELE value="FS_COLLECT" displayname="fs.Collection" />
<ELE value="FS_EXTRACT" displayname="fs.Extraction" />
</OPTION>
- <OPTION name="fsfilename">
<ELE value="false" displayname="fs.useOriginal" />
<ELE value="true" displayname="fs.assignName" />
</OPTION>
</SERVICES>
```

Filesystem_en File

The following sample code is a Filesystem_en file:

```
fs.label = File System Adapter
fs.description = Collects and Extracts files from a file system.
as2fs.label = AS2 File System Adapter
as2fs.description = Collects and Extracts as2 files from a file system.

fs.action = Action
fs.append = Append to file when extracting?
fs.delete = Delete file after collecting?
fs.zero = Collect zero byte files?
fs.modtime = The modification time of a file must be older than the number of seconds
specified or it will not be collected.
fs.cfolder = Collection folder
fs.efolder = Extraction folder
fs.filter = Filename filter
fs.pollinterval = Poll Interval (mins)
fs.subfolders = Collect files from sub folders within and including the collection
folder?
fs.keepPath = Use the absolute file path name for the document name?
fs.bootstrap = Start a business process once files are collected?
fs.extractfilename = Filename
fs.filename = Filenaming convention
fs.Collection = Collection
fs.Extraction = Extraction
fs.useOriginal = Use the original filename as the extracted filename
fs.assignName = Assign a specific name
fs.maxthreads = Maximum number of threads to use when bootstrapping collected files
fs.contract = Contratc

fs.wfd.group1.title = Workflow Properties
fs.wfd.group1.instructions = Specify the appropriate workflow settings.

fs.instance.group1.title = Collection
fs.instance.group1.instructions = \
<table border=0 cellpadding=1 cellspacing=1>\
<tr>\
  <td colspan=2 valign="top" align="left" class='info'>\
    Specify the appropriate settings for collecting data.\
  </td>\
</tr>\
<tr>\
  <TD class='bullet'>&#149;</TD>\
  <td valign="top" align="left" class='info'>\
    Specify the folder to collect files from.\
  </td>\
</tr>\
<tr>\
  <TD class='bullet'>&#149;</TD>\
  <td valign="top" align="left" class='info'>\
    An optional filter can be used to only collect certain files (i.e. *.txt).\
  </td>\
</tr>\
<tr>\
  <TD class='bullet'>&#149;</TD>\
  <td valign="top" align="left" class='info'>\
    Specify whether to search any sub folders within the main collection folder.\
  </td>\
</tr>\
<tr>\
  <TD class='bullet'>&#149;</TD>\
  <td valign="top" align="left" class='info'>\
    Specify whether or not a business process is started once files are collected.\
  </td>\
</tr>\
</table>
```

```
fs.instance.group2.title = Extraction
fs.instance.group2.instructions = \
<table border=0 cellpadding=1 cellspacing=1>\
<tr>\
  <td colspan=2 valign="top" align="left" class='info'>\
    Specify the appropriate settings for extracting files.\
  </td>\
</tr>\
<tr>\
  <TD class='bullet'>&#149;</TD>\
  <td valign="top" align="left" class='info'>\
    Specify the folder to extract files to.\
  </td>\
</tr>\
<tr>\
  <TD class='bullet'>&#149;</TD>\
  <td valign="top" align="left" class='info'>\
    Specify whether to use the original document filename or assign a different one.\
  </td>\
</tr>\
</table>

fs.instance.group2a.title = User defined
fs.instance.group2a.instructions = \
<table border=0 cellpadding=1 cellspacing=1>\
<tr>\
  <td colspan=2 valign="top" align="left" class='info'>\
    Specify the filename to assign to the extracted files.  To generate unique
filenames, the escape code %^ will be replaced with a unique number in the format
yyyymmddhhmmsslll (i.e. specifying Rcv%^.dat will generate Rcv200110191259123.dat).\
  </td>\
</tr>\
</table>
```

FileSystemImpl File

The following sample code is a FileSystemImpl file:

```
package com.sterlingcommerce.woodstock.services.filesystem;

import java.io.File;
import java.util.Arrays;
import com.sterlingcommerce.woodstock.util.Util;
import com.sterlingcommerce.woodstock.workflow.WorkFlowDef;
import com.sterlingcommerce.woodstock.workflow.WorkFlowContext;
import com.sterlingcommerce.woodstock.workflow.WorkFlowException;
import com.sterlingcommerce.woodstock.services.IService;
import com.sterlingcommerce.woodstock.services.XLogger;
import com.sterlingcommerce.woodstock.util.frame.Manager;
import com.sterlingcommerce.woodstock.util.frame.log.LogService;
import com.sterlingcommerce.woodstock.util.frame.lock.LockManager;
import com.sterlingcommerce.woodstock.services.controller.ServicesControllerImpl;
import org.w3c.dom.Document;
import org.w3c.dom.Node;

public class FileSystemImpl implements IService {
    static final int DEF_LOCKLOOP = 5000;
    static final int DEF_AGELOOP = 6;


//##################################################################################
############
    public WorkFlowContext processData(WorkFlowContext wfc) throws WorkFlowException
{
        String badCfg = "No 'Action' or request specified";
        String svcName = wfc.getServiceName();
        XLogger log = new XLogger("FileSystemImpl", svcName);
        ServicesControllerImpl sci = ServicesControllerImpl.getInstance();
        sci.harnessRegister(String.valueOf(wfc.getWorkFlowId()), svcName);
        try {
            FileSystemServer rmi = (FileSystemServer)sci.getAdapter(svcName);
            if ( rmi == null ) {
                handleError(wfc, log, "RMI instance is null");
            }
            else {
                wfc.suspendTransaction();
                String action = wfc.getParm("Action"); // determine action
                if ( action == null ) {
                    Node node = (Node)wfc.getWFContent("/*");
                    if ( node != null ) {
                        String inputMsg = node.getNodeName();
                        if ( inputMsg != null ) {
                            if ( inputMsg.equals("importFileRequest") ) {
                                doImportFile(wfc, log, rmi);
                            }
                            else if ( inputMsg.equals("exportDocumentRequest") ) {
                                doExportFile(wfc, log, rmi);
                            }
                            else { // no idea what to do
                                handleError(wfc, log, badCfg);
                            }
                        }
                        else { // inputMsg is null
                            handleError(wfc, log, badCfg);
                        }
                    }
                    else {  // node is null
                        handleError(wfc, log, badCfg);
                    }
                }
                else if ( action.equals("FS_EXTRACT") ) {
                    doExtract(wfc, log, rmi);
                }
                else if ( action.equals("FS_COLLECT") ) {
                    doCollect(wfc, log, rmi, svcName);
                }
                else {
                    handleError(wfc, log, "Unknown 'Action' value: " + action);
                }
                wfc.resumeTransaction();
            }
        }
        catch(Exception e) {
            handleException(wfc, log, "Exception in processData", e);
        }
        sci.unregisterThread();
        return wfc;
    }


//##################################################################################
############
    private void doExtract(WorkFlowContext wfc, XLogger log, FileSystemServer rmi) {
        String msg = "Exception in doExtract"; // a multi-use message string var
        try {
            String advStatus = null;
            FSToExtractInfo tei = new FSToExtractInfo();
            tei.folder = wfc.getParm("extractionFolder");
            if ( tei.folder == null || tei.folder.length() == 0 ) {
                handleError(wfc, log, "Missing required 'extractionFolder'
parameter");
            }
            else {
                com.sterlingcommerce.woodstock.workflow.Document doc =
wfc.getPrimaryDocument();
                if ( doc == null ) {
                    handleError(wfc, log, "Primary document is null");
                }
                else {
                    tei.data = doc.getBody();
                    if ( tei.data == null ) {
                        advStatus = "0 byte file";
                    }
                    tei.bodyName = doc.getBodyName();
                    if ( tei.bodyName == null || tei.bodyName.length() == 0 ) {
                        tei.bodyName = Util.getUniqueFileName("%^.dat");
                    }
                    if ( "true".equals(wfc.getParm("assignFilename")) ) {
                        tei.assignedFilename = wfc.getParm("assignedFilename");
                    }
                    if ( "true".equals(wfc.getParm("appendOnExtract")) ) {
                        tei.append = true;
                    }
                  msg = "Exception in writeFile"; // load in case it throws exception
                    rmi.writeFile(tei);
                    setStatus(wfc, WorkFlowContext.SUCCESS, advStatus);
                }
            }
        }
        catch(Exception e) {
            handleException(wfc, log, msg, e);
        }
    }
```

```
//##################################################################################
############
    private void doCollect(WorkFlowContext wfc, XLogger log, FileSystemServer rmi,
String svcName) {
        try {
            FSToCollectInfo tci = new FSToCollectInfo();
            tci.wfctm = wfc.packMessageToChild();
            tci.svcName = svcName;
            String tmp = wfc.getParm("fileModTimeThreshold");
            if ( tmp == null || tmp.length() == 0 ) {
                tmp = Manager.getProperty("FSAdapterFileModSeconds");
            }
            try {
                tci.modTime = Integer.parseInt(tmp);
            }
            catch(Exception e) {
                tci.modTime = 30;
            }
            tci.modTime *= 1000; // adjust to milliseconds

            tci.folder = wfc.getParm("collectionFolder");
            if ( tci.folder == null || tci.folder.length() == 0 ) {
                handleError(wfc, log, "Missing required 'collectionFolder'
parameter");
            }
            else {
                tmp = wfc.getParm("useSubFolders");
                if ( "true".equals(tmp) ) {
                    tci.useSubFolders = true;
                }
                tmp = wfc.getParm("keepPath");
                if ( "true".equals(tmp) ) {
                    tci.keepPath = true;
                }
                tmp = wfc.getParm("deleteAfterCollect");
                if ( "false".equals(tmp) ) {
                    tci.delete = false;
                }
                tmp = wfc.getParm("collectZeroByteFiles");
                if ( "true".equals(tmp) ) {
                    tci.getzero = true;
                }
                tci.filter = wfc.getParm("filter");
                if ( tci.filter == null || tci.filter.length() == 0 ) {
                    tci.filter = "*";
                }
                tmp = wfc.getParm("bootstrap");
                if ( "false".equals(tmp) ) {
                    doCollectNoBootstrap(wfc, log, rmi, tci);
                }
                else {
                    doCollectWithBootstrap(wfc, log, rmi, tci);
                }
            }
        }
        catch(Exception e) {
            handleException(wfc, log, "Exception in doCollect", e);
        }
    }


//##################################################################################
############
    private void doCollectWithBootstrap(WorkFlowContext wfc, XLogger log,
FileSystemServer rmi, FSToCollectInfo tci) {
        String msg = "Exception in doCollectWithBootstrap";
        boolean isLocked = false;
        try {
            String iwfId = wfc.getParm("initialWorkFlowId");
            String iwfName = wfc.getParm("initialWorkFlowName");
            if ( iwfId == null ) {
                if ( iwfName != null ) {
                    try {
                        tci.iwfid = WorkFlowDef.getIDForName(iwfName);
                    }
                    catch(Exception e) {
                        tci.iwfid = -1;
                    }
                }
            }
            else {
                try {
                    tci.iwfid = WorkFlowDef.getIDForName(iwfId);
                }
                catch(Exception e) {
                    try {
                        tci.iwfid = Integer.parseInt(iwfId);
                    }
                    catch(NumberFormatException nfe) {
                        tci.iwfid = -1;
                    }
                }
            }
            if ( tci.iwfid == -1 ) {
                handleError(wfc, log, "Required 'initialWorkFlowId' parameter not
found or invalid");
            }
            else {
                try {
                    tci.maxThreads = Integer.parseInt(wfc.getParm("maxThreads"));
                }
                catch(Exception e) {
                    tci.maxThreads = 10;
                }
                if ( tci.maxThreads < 1 ) {
                    tci.maxThreads = 1; // gotta have at least one
                }
                if ( LockManager.isLocked(tci.folder) ) {
                  setStatus(wfc, WorkFlowContext.SUCCESS, tci.folder + " is locked");
                }
                else {
                    LockManager.lock(tci.folder, tci.svcName, 0, true);
                    isLocked = true;
                    msg = "Exception in collect";
                    FSFromCollectInfo fci = rmi.collect(tci);
                    if ( fci.workflows != null && fci.filesCollected > 0 ) {
                        wfc.addBootStrapWorkFlows(fci.workflows);
                    }
                    LockManager.unlock(tci.folder);
                    isLocked = false;
                    if ( fci.couldntRead ) {
                        if ( fci.filesCollected == 0 ) {
                            handleError(wfc, log, "Unable to collect any files");
                        }
                        else {
                          setStatus(wfc, WorkFlowContext.SUCCESS, "Unable to collect
one or more files");
                        }
                    }
                    else if ( fci.filesCollected == 0 ) {
                      setStatus(wfc, WorkFlowContext.SUCCESS, "No files to collect");
                    }
                    else {
                        setStatus(wfc, WorkFlowContext.SUCCESS, fci.filesCollected +
" files collected");
                    }
                }
            }
        }
        catch(Exception e) {
            if ( isLocked ) {
                LockManager.unlock(tci.folder);
            }
            handleException(wfc, log, msg, e);
        }
    }


//#####################################################################################
############
    private void doCollectNoBootstrap(WorkFlowContext wfc, XLogger log,
FileSystemServer rmi, FSToCollectInfo tci) {
        String msg = "Exception in doCollectNoBootstrap";
        boolean isLocked = false;
        try {
            if ( tci.filter.indexOf("*") != -1 ) {
                int lockLoop = 5;
                while ( lockLoop-- > 0 ) {
                    if ( LockManager.isLocked(tci.folder) ) {
                        if ( lockLoop == 0 ) {
                          handleError(wfc, log, tci.folder + " is still locked after
5 attempts");
                            return;
                        }
                        try {
                            Thread.sleep(DEF_LOCKLOOP);
                        }
                        catch(Exception e) {
                            log.logException("Exception during Thread.sleep in lock
loop", e);
                        }
                    }
                    else {
                        break;
                    }
                }
                LockManager.lock(tci.folder, tci.svcName, 0, true);
                isLocked = true;
            }
            int filesRead = 0; // used when all reads return null so we can set "no
files to collect"
            boolean couldntRead = false;
            msg = "Exception in scanFolder";
          String[] files = rmi.scanFolder(tci.folder, tci.filter, tci.useSubFolders,
true); // get list of files
            if ( files != null ) {
                FSFromReadFile frf = null;
                int i = 0;
                int aged = DEF_AGELOOP;
                while ( i < files.length ) {
                    msg = "Exception in readFile";
                    frf = rmi.readFile(files[i], tci.modTime, true);
                    if ( frf.goodfile ) {
                        if ( frf.data == null ) {
                            if ( tci.getzero ) {
                             frf.data = "".getBytes(); // allows for collecting zero
byte files
                            }
                            else {
                                i++;
                                continue;
                            }
                        }
                        filesRead++;
                        com.sterlingcommerce.woodstock.workflow.Document doc =
wfc.createDocument();
                        doc.setBody(frf.data);
                        String tmp = files[i];
                        if ( !tci.keepPath ) {
                            tmp = new File(files[i]).getName();
                        }
                        doc.setBodyName(tmp);
                        wfc.putPrimaryDocument(doc);
                        if ( tci.delete ) {
                            msg = "Exception in deleteFile";
                            rmi.deleteFile(files[i]);
                        }
                        break;
                    }
                    else if ( frf.data != null && Arrays.equals(frf.data,
FileSystemServerImpl.CANTREAD) ) {
                        couldntRead = true;
                    }
                    else if ( files.length == 1 && aged-- > 0 ) {
                       // if we get to this point, we're trying to read just one file
and it wasn't aged enough
                        try {
                            Thread.sleep(tci.modTime);
                        }
                        catch(Exception e) {
                            log.logException("Exception during Thread.sleep", e);
                        }
                        continue; // won't increment so it will try getting the same
file again
                    }
                    i++;
                }
            }
            if ( couldntRead ) {
                if ( filesRead == 0 ) {
                    handleError(wfc, log, "Unable to collect any files");
                }
                else {
                    setStatus(wfc, WorkFlowContext.SUCCESS, null);
                }
            }
            else if ( filesRead == 0 ) {
                handleError(wfc, log, "No files to collect");
            }
            else {
                setStatus(wfc, WorkFlowContext.SUCCESS, null);
            }
        }
        catch(Exception e) {
            handleException(wfc, log, msg, e);
        }
        finally {
            if ( isLocked ) {
                LockManager.unlock(tci.folder);
            }
        }
    }


//##################################################################################
############
    private void doImportFile(WorkFlowContext wfc, XLogger log, FileSystemServer rmi)
{
        String msg = "Exception in doImportFile";
        try {
            if ( LogService.out.debug ) {
                log.logDebug("importFileRequest");
            }
            String filename = wfc.getParm("filename"); // ImportFile
            if ( filename == null || filename.length() == 0 ) {
                handleError(wfc, log, "Missing required 'filename' parameter");
            }
            else {
                String dirname = wfc.getParm("dirname");
                FSFromReadFile frf = null;
                int aged = DEF_AGELOOP;
                while ( aged-- > 0 ) {
                    msg = "Exception in importFile";
                    frf = rmi.importFile(dirname, filename);
                    if ( frf.goodfile ) {
                        if ( frf.data == null ) {
                            frf.data = "".getBytes(); // allows for collecting zero
byte files
                        }
                        com.sterlingcommerce.woodstock.workflow.Document doc =
wfc.createDocument();
                        doc.setBody(frf.data);
                        doc.setBodyName(filename);
                        wfc.putDocument("document", doc);
                        Object value = doc.getDocumentId();
                        if ( value instanceof org.w3c.dom.Document ) {
                          value = ((org.w3c.dom.Document)value).getDocumentElement();
                        }
                        wfc.setWFContent("doc:document-id", value, false);
                        setStatus(wfc, WorkFlowContext.SUCCESS, null);
                        aged = 1; // just in case it's zero at the time it works
                        break;
                    }
                    else if ( frf.data != null && Arrays.equals(frf.data,
FileSystemServerImpl.CANTREAD) ) {
                        handleError(wfc, log, "Unable to read " + frf.importFile);
                    }
                    else {
                        // if we get here, the file is probably not aged enough
                        try {
                            Thread.sleep(FileSystemServerImpl.DEF_MODTIME);
                        }
                        catch(Exception e) {
                            log.logException("Exception during Thread.sleep", e);
                        }
                    }
                }
                if ( aged == 0 ) { // if we get here - importFileRequest did not work
                    handleError(wfc, log, frf.importFile + " was not collected");
                }
            }
        }
        catch(Exception e) {
            handleException(wfc, log, msg, e);
        }
    }


//####################################################################################
############
    private void doExportFile(WorkFlowContext wfc, XLogger log, FileSystemServer rmi)
{
        String msg = "Exception in doExportFile";
        try {
            if ( LogService.out.debug ) {
                log.logDebug("exportDocumentRequest");
            }
            String docId = wfc.getParm("doc:document-id");
            String filename = wfc.getParm("filename");
            String prefix = wfc.getParm("filename-prefix");
            if ( docId == null || docId.trim().length() == 0 ) {
               handleError(wfc, log, "Missing required 'doc:document-id' parameter");
            }
            else {
                if ( prefix == null || prefix.trim().length() == 0 ) {
                    prefix = "document-";
                }
                if ( filename == null || filename.trim().length() == 0 ) {
                    filename = prefix + docId;
                }
                com.sterlingcommerce.woodstock.workflow.Document doc = new
com.sterlingcommerce.woodstock.workflow.Document(docId);
                if ( doc == null ) {
                    handleError(wfc, log, "Document is null: " + docId);
                }
                else {
                    msg = "Exception in exportFile";
                    rmi.exportFile(filename, doc.getBody());
                    wfc.setWFContent("filename", filename);
                    setStatus(wfc, WorkFlowContext.SUCCESS, null);
                }
            }
        }
        catch(Exception e) {
            handleException(wfc, log, msg, e);
        }
    }


//##################################################################################
############
    private void handleError(WorkFlowContext wfc, XLogger log, String advStatus) {
        log.logError(advStatus);
        setStatus(wfc, WorkFlowContext.ERROR, advStatus);
    }


//##################################################################################
############
    private void handleException(WorkFlowContext wfc, XLogger log, String advStatus,
Exception e) {
        log.logException(advStatus, e);
        setStatus(wfc, WorkFlowContext.ERROR, e.getMessage());
    }


//##################################################################################
############
    private void setStatus(WorkFlowContext wfc, int basicStatus, String advStatus) {
        wfc.setBasicStatus(basicStatus);
        if ( advStatus != null ) {
            wfc.setAdvancedStatus(advStatus);
```

FileSystemServer File

The following sample code is a FileSystemServer file:

```
package com.sterlingcommerce.woodstock.services.filesystem;

import java.rmi.*;
import java.rmi.server.*;
import java.rmi.RemoteException;
import com.sterlingcommerce.woodstock.services.AdapterException;
import com.sterlingcommerce.woodstock.services.IAdapterRMI;

/**
 * Remote interface of the I/O routines used by the File System Adapter
 * @since Woodstock 2.0
 */
public interface FileSystemServer extends IAdapterRMI
{
    public FSFromCollectInfo collect(FSToCollectInfo tci) throws AdapterException,
RemoteException;
    public String[] scanFolder(String folder, String fileFilter, boolean
useSubFolders, boolean doreg) throws AdapterException, RemoteException;
    public FSFromReadFile readFile(String absoluteFileName, int modTime, boolean
doreg) throws AdapterException, RemoteException;
    public FSFromReadFile importFile(String dirname, String filename) throws
AdapterException, RemoteException;
    public void exportFile(String filename, byte[] data) throws AdapterException,
RemoteException;
    public void writeFile(FSToExtractInfo tei) throws AdapterException,
RemoteException;
    public void deleteFile(String absoluteFileName) throws AdapterException,
RemoteException;
}
```

FileSystemServerImpl File

The following sample code is a FileSystemServerImpl file:

```
package com.sterlingcommerce.woodstock.services.filesystem;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.BufferedInputStream;
import java.io.BufferedOutputStream;
import java.io.ByteArrayOutputStream;
import java.util.Properties;
import java.util.Vector;
import com.sterlingcommerce.woodstock.util.Util;
import com.sterlingcommerce.woodstock.util.WildCardFilter;
import com.sterlingcommerce.woodstock.services.AdapterException;
import com.sterlingcommerce.woodstock.services.IAdapterImpl;
import com.sterlingcommerce.woodstock.util.frame.Manager;
import com.sterlingcommerce.woodstock.util.frame.log.LogService;

public class FileSystemServerImpl extends IAdapterImpl implements FileSystemServer {
    public FileSystemServerImpl() { super(); }
    public void refreshAdapter(Properties p) {}
    public String message(String s) { return s; }
    public void startupAdapter(Properties p) {}
    public void shutdownAdapter() {}
    public static final byte[] CANTREAD = "<Can't @#$ read>".getBytes(); // special
chars used to make it unique (encoding doesn't matter)
    public static final int DEF_MODTIME = 30000; // 30 seconds
    private static final int BUFFER_SIZE = 1024;

    /**
     * Method for collecting files and starting workflows
     * @param tci a FSToCollectInfo class containing all the necessary parms
     * @return a FSFromCollectInfo class containing all the necessary info
     * @since Woodstock 2.0
     */
    public FSFromCollectInfo collect(FSToCollectInfo tci) throws AdapterException {
        registerThread();
        FSFromCollectInfo fci = new FSFromCollectInfo();
        WFStartThread wfst = null;
        Thread t = null;
        try {
            String[] files = scanFolder(tci.folder, tci.filter, tci.useSubFolders,
false);
            if ( files != null ) {
                fci.workflows = new Vector(files.length);
                for ( int i = 0; i < files.length; i++ ) {
                    while ( fci.threadCnt == tci.maxThreads ) {
                        Thread.yield();
                    }
                    wfst = new WFStartThread(files[i], tci, fci, this);
                    t = new Thread(wfst);
                    fci.threadCounter(true); // synchronized increment
                    t.start();
                }
                while ( fci.threadCnt > 0 ) {
                    Thread.yield();
                }
            }
        }
        catch(Exception e) {
            LogService.out.logException("Exception in collect method", e);
            unregisterThread();
            throw new AdapterException(e);
        }
        unregisterThread();
        return fci;
    }

    /**
```

```
     * Method for returning an array of filenames in a directory
     * @param folderName name of the directory
     * @param fileFilter file filter to apply
     * @param useSubFolders whether to scan subdirectories or not
    * @param doreg controls whether it was called from the collect method internally
or from a RMI call
     * @return A string array of filenames
     * @since Woodstock 2.0
     */
    public String[] scanFolder(String folderName, String fileFilter, boolean
useSubFolders, boolean doreg) throws AdapterException {
        if ( doreg ) {
            registerThread();
        }
        String folder = null;
        try {
            folder = new File(folderName).getCanonicalPath();
        }
        catch(Exception e) {
           folder = folderName; // should never happen but in case it does, just use
it as is
        }
        File wdir = new File(folder);
        if ( !wdir.exists() ) {
            throw new AdapterException(folder + " does not exist");
        }

        Vector fileVect = new Vector(10);
        WildCardFilter filter = new WildCardFilter(fileFilter);
        traverseDir(fileVect, filter, folder, useSubFolders);
        String[] fileNames = null;
        int vectSize = fileVect.size();
        if ( vectSize > 0 ) {
            fileNames = new String[vectSize];
            while ( vectSize-- > 0 ) {
                fileNames[vectSize] = (String)fileVect.elementAt(vectSize);
            }
        }
        if ( doreg ) {
            unregisterThread();
        }
        return fileNames;
    }

    /**
     * Method for recursively traversing a directory structure
     * @param fileVect Vector object used to collect the recursed information
     * @param filter file filter to apply
     * @param folderName name of the directory
     * @param useSubFolders whether to scan subdirectories or not
     * @since Woodstock 2.0
     */
    public void traverseDir(Vector fileVect, WildCardFilter filter, String
folderName, boolean useSubFolders) {
        if ( folderName != null ) {
            File folder = new File(folderName);
            File[] fileList = folder.listFiles();
            if ( fileList != null ) {
                for ( int i = 0; i < fileList.length; i++ ) {
                    if ( !fileList[i].isDirectory() ) {
                        if ( filter.accept(folder, fileList[i].getName()) ) {
                            try {
                                fileVect.add(fileList[i].getCanonicalPath());
                            }
                            catch(Exception e) {
                                fileVect.add(fileList[i].getAbsolutePath());
                            }
                        }
                    }
                    else if ( useSubFolders ) {
                        try {
                            traverseDir(fileVect, filter,
fileList[i].getCanonicalPath(), useSubFolders);
                        }
                        catch(Exception e) {
                            traverseDir(fileVect, filter,
fileList[i].getAbsolutePath(), useSubFolders);
                        }
                    }
                }
            }
        }
    }

    /**
     * Method for reading a file from disk.<p>
     * @param absoluteFileName name of the file (inc. its absolute path name)
     * @param modTime the value used to check against the file modification time to
determine whether or not to pickup
     * @param doreg controls whether it was called from the collect method internally
or from a RMI call
     * @return FSFromReadFile information passed back from this method
     * @exception AdapterException
     * @since Woodstock 2.0
     */
    public FSFromReadFile readFile(String absoluteFileName, int modTime, boolean
doreg) throws AdapterException {
        if ( doreg ) {
            registerThread();
        }
        FSFromReadFile frf = new FSFromReadFile();
        if ( LogService.out.debug ) {
            LogService.out.logDebug("Trying to read file " + absoluteFileName);
        }
        File file = new File(absoluteFileName);
        if ( !file.canRead() ) {
            if ( LogService.out.debug ) {
                LogService.out.logDebug("can't read file");
            }
            if ( doreg ) {
                unregisterThread();
            }
            frf.data = CANTREAD;
            return frf;
        }
        final long lastmod = file.lastModified();
        final long curtime = System.currentTimeMillis();
        if ( curtime - lastmod < modTime ) {
            if ( LogService.out.debug ) {
                LogService.out.logDebug("Modification time still under limit");
            }
            if ( doreg ) {
                unregisterThread();
            }
            return frf;
        }
        final int fileSize = (int)file.length();
        int bytesRead = 0;
        int bufSize = BUFFER_SIZE;
        ByteArrayOutputStream baoStream = new ByteArrayOutputStream(fileSize);
        BufferedInputStream biStream = null;

        if ( fileSize > 0 ) {
            try {
                if ( fileSize < bufSize ) {
                    bufSize = fileSize;
                }
                biStream = new BufferedInputStream(new FileInputStream(file),
bufSize*2);
                frf.data = new byte[bufSize];

                while ( (bytesRead = biStream.read(frf.data)) != -1 ) {
                    baoStream.write(frf.data, 0, bytesRead);
                }
                frf.data = baoStream.toByteArray();
            }
            catch(Exception e) {
                if ( doreg ) {
                    unregisterThread();
                }
                LogService.out.logException("Exception in readFile:", e);
                throw new AdapterException(e);
            }
            finally {
                if ( biStream != null ) {
                    try {
                        biStream.close();
                    }
                    catch(Exception e) {
                        LogService.out.logWarn(e.getMessage());
                    }
                }
            }
        }

        // check if file has been modified since read
        if ( file.lastModified() == lastmod ) {
            frf.goodfile = true;
        }
        if ( doreg ) {
            unregisterThread();
        }
        return frf;
    }

    /**
     * Method for writing a file to disk.<p>
     * @param tei a FSToExtractInfo class containing all the necessary parms
     * @exception AdapterException
     * @since Woodstock 2.0
     */
    public void writeFile(FSToExtractInfo tei) throws AdapterException {
        registerThread();
        BufferedOutputStream boStream = null;
        try {
            String filename = null;
            String folder = new File(tei.folder).getCanonicalPath();
            if ( tei.assignedFilename != null && tei.assignedFilename.length() > 0 ) {
                tei.assignedFilename = new File(tei.assignedFilename).getName();
                filename = folder + java.io.File.separator +
Util.getUniqueFileName(tei.assignedFilename);
            }
            else {
                filename = folder + java.io.File.separator + new
File(tei.bodyName).getName();
            }
            boStream = new BufferedOutputStream(new FileOutputStream(filename,
tei.append));
            if ( tei.data != null ) {
                boStream.write(tei.data, 0, tei.data.length);
                boStream.flush();
            }
            boStream.close();

            // Workaround for websphere bug
            String rfp = Manager.getProperty("resetFilePerms");
            if ( "true".equalsIgnoreCase(rfp) || "yes".equalsIgnoreCase(rfp) ) {
                String mask = Manager.getProperty("resetFilePermsMask");
                if ( mask == null ) {
                    mask = "664";
                }
                String os = System.getProperty("os.name");
                if ( os != null && os.toLowerCase().indexOf("windows") == -1 ) {
                    Runtime.getRuntime().exec(new String[]{"chmod", mask, filename});
                }
            }
        }
        catch(Exception e) {
            unregisterThread();
            throw new AdapterException(e);
        }
        finally {
            if ( boStream != null ) {
                try {
                    boStream.flush();
                    boStream.close();
                }
                catch(Exception e) {
                    LogService.out.logWarn(e.getMessage());
                }
            }
        }
        unregisterThread();
    }

    /**
     * Method for deleting a file on disk
     * @param absoluteFileName name of the file with its absolute path
     * @exception AdapterException
     * @since Woodstock 2.0
     */
     public void deleteFile(String absoluteFileName) throws AdapterException {
        registerThread();
        File file = null;
        try {
            file = new File(absoluteFileName);
            file.delete();
        }
        catch(Exception e) {
            unregisterThread();
            throw new AdapterException(e);
        }
        unregisterThread();
     }

    /**
     * Method used to import a file directly
     * @param dirname the directory name
     * @param filename the filename
     * @return FSFromReadFile information passed back from this method
     * @since Woodstock 2.2
     */
    public FSFromReadFile importFile(String dirname, String filename) throws
AdapterException {
        registerThread();
        String fileToRead = filename;
        if ( dirname != null && dirname.length() > 0 ) {
            try {
                fileToRead = new File(dirname, filename).getCanonicalPath();
            }
            catch(Exception e) {
                unregisterThread();
                throw new AdapterException(e);
            }
        }

        FSFromReadFile frf = null;
        try {
            frf = readFile(fileToRead, DEF_MODTIME, false);
        }
        catch(Exception e) {
            unregisterThread();
            throw new AdapterException(e);
        }
        frf.importFile = fileToRead;
        unregisterThread();
        return frf;
    }

    /**
     * Method used to export a file directly
     * @param filename the filename
     * @since Woodstock 2.2
     */
    public void exportFile(String filename, byte[] data) throws AdapterException {
        registerThread();
        BufferedOutputStream boStream = null;
        try {
            boStream = new BufferedOutputStream(new FileOutputStream(filename));
            if ( data != null ) {
                boStream.write(data, 0, data.length);
                boStream.flush();
            }
            boStream.close();

            // Workaround for websphere bug
            String rfp = Manager.getProperty("resetFilePerms");
            if ( "true".equalsIgnoreCase(rfp) || "yes".equalsIgnoreCase(rfp) ) {
                String mask = Manager.getProperty("resetFilePermsMask");
                if ( mask == null ) {
                    mask = "664";
                }
                String os = System.getProperty("os.name");
                if ( os != null && os.toLowerCase().indexOf("windows") == -1 ) {
                    Runtime.getRuntime().exec(new String[]{"chmod", mask, filename});
                }
            }
        }
        catch(Exception e) {
            unregisterThread();
            throw new AdapterException(e);
        }
        finally {
            if ( boStream != null ) {
                try {
                    boStream.flush();
                    boStream.close();
                }
                catch(Exception e) {
                    LogService.out.logWarn(e.getMessage());
                }
            }
        }
        unregisterThread();
    }
}
```

FSFromCollectInfo File

The following sample code is an FSFromCollectInfo file:

```
package com.sterlingcommerce.woodstock.services.filesystem;

import java.util.Vector;

/**
 * Information passed from the collect method
 * @since Woodstock 2.1
 */
public class FSFromCollectInfo implements java.io.Serializable {
    public int threadCnt = 0;
    public int filesCollected = 0;
    public boolean couldntRead = false;
    public Vector workflows = null;

    public FSFromCollectInfo() {}

    public synchronized void addBootstrapWF(String wfId) {
        workflows.add(wfId);
    }

    public synchronized void fileCounter() {
        filesCollected++;
    }

    public synchronized void threadCounter(boolean add) {
        if ( add ) {
            threadCnt++;
        }
        else {
            threadCnt--;
```

FSFromReadFile File

The following sample code is an FSFromReadFile:

```
package com.sterlingcommerce.woodstock.services.filesystem;

/**
 * Information passed from the readFile method
 * @since Woodstock 2.1
 */
public class FSFromReadFile implements java.io.Serializable
{
    byte[] data = null;
    boolean goodfile = false;
    String importFile = null;

    public FSFromReadFile() {}
}
```

FSToCollectInfo File

The following sample code is an FSToCollectInfo file:

```
package com.sterlingcommerce.woodstock.services.filesystem;

import java.util.Vector;
import com.sterlingcommerce.woodstock.workflow.WFCTransportMessage;

/**
 * Information passed to the collect method
 * @since Woodstock 2.1
 */
public class FSToCollectInfo implements java.io.Serializable {
    public int iwfid = -1;
    public int modTime = 0;
    public int maxThreads = 10;
    public boolean useSubFolders = false;
    public boolean keepPath = false;
    public boolean delete = true;
    public boolean getzero = false;
    public String folder = null;
    public String filter = null;
    public String svcName = null;
    public WFCTransportMessage wfctm = null;
    public FSToCollectInfo() {}
}
```

FSToExtractInfo File

The following sample code is an FSToExtractInfo file:

```
package com.sterlingcommerce.woodstock.services.filesystem;

import java.util.Vector;

/**
 * Information passed to the collect method
 * @since Woodstock 2.1
 */
public class FSToExtractInfo implements java.io.Serializable
{
    public boolean append = false;
    public String folder = null;
    public String bodyName = null;
    public String assignedFilename = null;
    public byte[] data = null;

    public FSToExtractInfo() {}
}
```

WFStartThread File

The following sample code is a WFStartThread file:

```
package com.sterlingcommerce.woodstock.services.filesystem;

import java.io.File;
import java.util.Arrays;
import com.sterlingcommerce.woodstock.services.XLogger;
import com.sterlingcommerce.woodstock.workflow.InitialWorkFlowContext;
import com.sterlingcommerce.woodstock.workflow.WorkFlowContextCookie;

class WFStartThread implements Runnable {
    private String filename = null;
    private FSToCollectInfo tci = null;
    private FSFromCollectInfo fci = null;
    private FileSystemServerImpl parentThread = null;

   public WFStartThread(String filename, FSToCollectInfo tci, FSFromCollectInfo fci,
FileSystemServerImpl fsImpl) {
        this.filename = filename;
        this.tci = tci;
        this.fci = fci;
        parentThread = fsImpl;
    }
    public void run() {
        String msg = "Exception reading file"; // initialize to first possible
exception
        try {
            FSFromReadFile frf = parentThread.readFile(filename, tci.modTime, false);
            if ( frf.goodfile ) {
                if ( frf.data == null && tci.getzero ) {
                   frf.data = "".getBytes(); // allows for collecting zero byte files
                }
                // don't do an 'else' here because frf.data might have just been set
above
                if ( frf.data != null ) { // only collect if data not null at this
point
                    String file2del = filename; // in case path is stripped off
                    if ( !tci.keepPath ) {
                        filename = new File(filename).getName();
                    }
                    InitialWorkFlowContext iwfc = new InitialWorkFlowContext();
                    iwfc.addContentElement("FileName", filename);
                    iwfc.setWorkFlowDefId(tci.iwfid);
                    iwfc.setDocumentName(filename);
                    iwfc.setDocumentBody(frf.data);
                    iwfc.setMessageToChild(tci.wfctm);
                    msg = "Exception starting workflow";
                    WorkFlowContextCookie wfcCookie = iwfc.start();
                    if ( wfcCookie != null ) {
                       fci.addBootstrapWF(Long.toString(wfcCookie.getWorkFlowId()));
                    }
                    fci.fileCounter(); // synchronized increment

                    if ( tci.delete ) {
                        msg = "Exception deleting file";
                        File file = new File(file2del);
                        file.delete();
                    }
                }
            }
            else if ( frf.data != null && Arrays.equals(frf.data,
FileSystemServerImpl.CANTREAD) ) {
                fci.couldntRead = true;
            }
        }
        catch(Exception e) {
            XLogger log = new XLogger("WFStartThread", tci.svcName);
            log.logException(msg, e);
        }
        finally {
            fci.threadCounter(false); // synchronized decrement
```

Index