Banner Workflow 8.0
Technical Integration Guide

Release 8.0
January 2009 (Revised)
## Section 1 Installation

### Chapter 1 Banner Workflow Installation

- **Hardware and Software Requirements** ........................................... 16
- **Banner Workflow - Client** ........................................................... 16
- **Banner Workflow - Server** ........................................................... 16
- **Required Software** ......................................................................... 16
- **Required Hardware** ....................................................................... 17

#### Installing Oracle Application Server: Oracle Containers for J2EE ............................................. 18

- **Setting Up Java** ............................................................................ 18

#### CD Directory Structure ................................................................. 18

- **Installation CD** ........................................................................... 19
- **Directory Structure** ..................................................................... 19
  - WORKFLOW_HOME ..................................................................... 19

#### Create and Initialize Installation Directory .............................................. 20

- **Installation** .................................................................................. 21
  - **Manually Edit configuration.xml** ................................................ 25
  - **Workflow Data Source wizard** .................................................... 26
  - **Banner Data Source wizard** ....................................................... 27
  - **Application Server wizard** ......................................................... 27
  - **Web Settings** ............................................................................ 28
  - **Workflow Engine** ...................................................................... 29
  - **Email** ....................................................................................... 31
  - **Banner Automated Activity Data Source** .................................... 31
  - **Banner Event Dispatcher** ............................................................. 31
  - **Root Account Password** ............................................................... 32
Re-deploying after configuration changes.  
Migrating from Workflow 4.4.2 to Banner Workflow 8.0  
Starting and Stopping Banner Workflow.  
Starting Banner Workflow  
Tip:  
Stopping Banner Workflow  
Supported Integration Modes for Banner Workflow  
Setting up Banner Integration  
Setting up Luminis Integration  
Configuring Banner Workflow without Banner  
Running the Workflow Engine as a Windows Service  
Log Files  
Deploying and Managing Multiple Engine Instances  
Overview  
Install an Engine Instance.  
Example.  
User Information That Is Extracted  

Chapter 2  
Banner Integration  

Banner Integration Overview  
Event Service  
Technologies to Launch Banner  
Banner Launcher Service  
Keeping Passwords in Sync between Banner Workflow and Banner  
Set Up Banner and Banner Workflow  
Enable Banner Workflow in GUAINST  
Enable the Event Queue  
Set Up Banner Workflow with Internet Native Banner  
Password Synchronization  
Set the Launch Service  
User Information That Is Extracted
Chapter 3  Integrating Banner Workflow with Luminis

Configuring Luminis for Workflow External Authentication ................. 76
Deploying the CAR File ................................................. 76
Configuring Single Sign On from Luminis to Banner Workflow ........... 77

Configuring Banner Workflow for Luminis SSO. ............................ 77
Luminis 4 Configuration .................................................. 78

Configuring Luminis for IDM or CAS Authentication ....................... 79
Deploying the CAR File ................................................. 79

Channel Administration ................................................ 80
Install the Worklist Channel ........................................... 80
Install the Workflow Alert Channel .................................. 81
Install the Workflow Processes Channel ................................ 82
Adding a Workflow Tab .................................................... 83

Section 2  Administration

Chapter 4  Server Administration

Server Time Synchronization .............................................. 87
Starting Banner Workflow ............................................... 87
<table>
<thead>
<tr>
<th>Contents</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stopping Banner Workflow</td>
<td>88</td>
</tr>
<tr>
<td>Import and Export Tools</td>
<td>88</td>
</tr>
<tr>
<td>Import a File</td>
<td>90</td>
</tr>
<tr>
<td>Export the Current Banner Workflow System</td>
<td>91</td>
</tr>
<tr>
<td>Modifying .ZIP Files</td>
<td>92</td>
</tr>
<tr>
<td>ZIP File Name</td>
<td>92</td>
</tr>
<tr>
<td>Relationship Between the XML Content File and the staticdocuments</td>
<td>92</td>
</tr>
<tr>
<td>Important Notes</td>
<td>92</td>
</tr>
<tr>
<td>Import File Manipulation with Castor Generated Classes</td>
<td>93</td>
</tr>
<tr>
<td>Extractor Tool</td>
<td>94</td>
</tr>
<tr>
<td>Archive and Purge</td>
<td>94</td>
</tr>
<tr>
<td>Archive Tool</td>
<td>94</td>
</tr>
<tr>
<td>Parameters</td>
<td>95</td>
</tr>
<tr>
<td>Archive Format</td>
<td>95</td>
</tr>
<tr>
<td>Purge Tool</td>
<td>96</td>
</tr>
<tr>
<td>Chapter 5 Database and Application Server Administration</td>
<td></td>
</tr>
<tr>
<td>Redeploying</td>
<td>99</td>
</tr>
<tr>
<td>configuration.xml</td>
<td>100</td>
</tr>
<tr>
<td>EmailServer</td>
<td>100</td>
</tr>
<tr>
<td>AutomatedActivityServlet</td>
<td>102</td>
</tr>
<tr>
<td>Reaper</td>
<td>102</td>
</tr>
<tr>
<td>EventDispatcher</td>
<td>102</td>
</tr>
<tr>
<td>DataSources</td>
<td>103</td>
</tr>
<tr>
<td>For Example.</td>
<td>104</td>
</tr>
<tr>
<td>What types of external data sources can be defined</td>
<td>105</td>
</tr>
<tr>
<td>SecurityIntegration</td>
<td>105</td>
</tr>
<tr>
<td>&lt;SuperUser&gt;</td>
<td>105</td>
</tr>
<tr>
<td>&lt;WebServicesUser&gt;</td>
<td>105</td>
</tr>
<tr>
<td>&lt;ExternalAuthentication&gt;</td>
<td>106</td>
</tr>
<tr>
<td>Engines</td>
<td>106</td>
</tr>
<tr>
<td>Overview</td>
<td>106</td>
</tr>
<tr>
<td>Deployment</td>
<td>107</td>
</tr>
<tr>
<td>WorkflowDataSource</td>
<td>108</td>
</tr>
<tr>
<td>BannerDataSource</td>
<td>108</td>
</tr>
<tr>
<td>Topic</td>
<td>Page</td>
</tr>
<tr>
<td>------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>ApplicationName</td>
<td>108</td>
</tr>
<tr>
<td>ApplicationServerHost</td>
<td>109</td>
</tr>
<tr>
<td>WebApplication</td>
<td>109</td>
</tr>
<tr>
<td>LuminisIntegration.</td>
<td>109</td>
</tr>
<tr>
<td>AdvancedDeployment.</td>
<td>110</td>
</tr>
<tr>
<td>Logging</td>
<td>110</td>
</tr>
<tr>
<td>Manipulating Data Sources through Banner Workflow</td>
<td>111</td>
</tr>
<tr>
<td>Connection Management</td>
<td>112</td>
</tr>
<tr>
<td>Automated Activity Deployment</td>
<td>114</td>
</tr>
<tr>
<td>Internal</td>
<td>114</td>
</tr>
<tr>
<td>Automated Workflow-aware and Non-workflow-aware</td>
<td>115</td>
</tr>
<tr>
<td>Clone a Banner Workflow Database</td>
<td>115</td>
</tr>
<tr>
<td>Chapter 6 Client Administration</td>
<td></td>
</tr>
<tr>
<td>Launch Services</td>
<td>117</td>
</tr>
<tr>
<td>SimpleClientLauncher</td>
<td>118</td>
</tr>
<tr>
<td>SimpleWebLauncher</td>
<td>118</td>
</tr>
<tr>
<td>WorkflowawareClientLauncher</td>
<td>118</td>
</tr>
<tr>
<td>Banner Launchers</td>
<td>118</td>
</tr>
<tr>
<td>Built-in Launch Parameters</td>
<td>119</td>
</tr>
<tr>
<td>Desktop Application Deployments</td>
<td>120</td>
</tr>
<tr>
<td>Java Web Start / Certificates</td>
<td>120</td>
</tr>
<tr>
<td>Signed Jars</td>
<td>120</td>
</tr>
<tr>
<td>Java Web Start with Internet Explorer</td>
<td>121</td>
</tr>
<tr>
<td>Java Web Start with Netscape Navigator</td>
<td>121</td>
</tr>
<tr>
<td>Desktop.sct</td>
<td>122</td>
</tr>
<tr>
<td>Internationalization</td>
<td>123</td>
</tr>
<tr>
<td>Look and Feel</td>
<td>123</td>
</tr>
</tbody>
</table>
# Chapter 10 Authentication

- **External Authentication** ............................................. 145
- Banner Authentication .................................................. 145
- LDAP Support ............................................................... 146
- Search ........................................................................... 146
- Security ........................................................................... 149
- **SSO Authentication** ..................................................... 149
- UDC ID Contract ............................................................. 150
- Configure Banner Workflow for CAS .................................. 150
- Configure IDM Gateway ................................................... 150
- Configure Banner Workflow for IDM Gateway ...................... 151
- Configure the `<LogoffUrl>` element .................................. 151
- Configure Luminis for Banner Workflow .............................. 151

# Chapter 11 Logging

- **Banner Workflow Logs** ................................................. 153
- Engine Logs ..................................................................... 153
  - WORKFLOW_HOME/engine/engine.log ............................... 153
  - WORKFLOW_HOME/engine/install.log .............................. 153
- **Oracle Application Server Logs** ..................................... 154
  - ORACLE_HOME/dcm/logs/dcmctl_logs/log.xml .................... 154
  - ORACLE_HOME/opmn/logs/<OC4J Instance Name>.default_island.1 154

# Chapter 12 Scripts

- archive ........................................................................... 155
- checkexternalauth ......................................................... 156
- configwizard ................................................................. 156
- export ............................................................................ 156
- extractwd ....................................................................... 157
- import ............................................................................. 158
Chapter 13 Tuning the Workflow Engine

Workflow Engine. ...................................................... 165
Interactive. ......................................................... 166
Asynchronous. ..................................................... 166

Engine Manager ..................................................... 167

How to Monitor Engine Performance. ......................... 168
ENG_EVENTS Table ................................................. 169

Chapter 14 Troubleshooting

Launching Banner from Banner Workflow .......................... 171
Processing Workflows ............................................. 171
Processing Business Events ....................................... 173
Customizing Example Workflows ................................ 173

Chapter 15 Security Management in Banner Workflow

Add a Role Authorization to an Existing Security Group ........ 175
Delete a Role Authorization from an Existing Security Group ... 176
Security Groups .................................................... 176
Appendix A  Banner Workflow Entity Relationship Diagrams

Archive .................................................. A-3
Calendar .................................................. A-4
Component ................................................ A-5
Component Query ...................................... A-6
Custom Activity ......................................... A-7
Document .................................................. A-8
Event ....................................................... A-8
Event Query ............................................. A-9
External Events ......................................... A-10
Process ..................................................... A-11
Process Definition ...................................... A-12
Process Definition Query .............................. A-13
Process Query ........................................... A-14
Role and User .......................................... A-15
Runtime Views .......................................... A-16
Workflow Engine ........................................ A-17
Workflow Engine Views ............................... A-18

Appendix B  SSL with Oracle Application Server

Using SSL with both Banner Workflow and Banner ........................................ B-3
Web Service Applications ................................ B-3
Appendix C  Altering Web Session Timeouts

Alter Web Session Timeouts ........................................... C-1

Appendix D  3rd Party Software Licenses

3rd Party Software Licenses ........................................... D-1
Section I

Installation
Before installing Banner Workflow 8.0, you should review the Hardware and Software Requirements section.

Banner Workflow 8.0 is compatible with specified versions of Banner. If you will be using Banner with Banner Workflow, Banner must be installed prior to installing Banner Workflow 8.0. The release of Banner Workflow 8.0 contains both a cumulative release of Banner Workflow 8.0 for new installations and scripts that can be used to easily upgrade existing Workflow 4.4.2 systems.

If you have not installed Banner Workflow:

- Banner Workflow 8.0 contains a cumulative release; you are not required to install a previous release of Banner Workflow before you install 8.0.

To migrate from a previous installation of Banner Workflow you must first migrate to Workflow 4.4.2 prior to migrating to Banner Workflow 8.0. See “Migrating from Workflow 4.4.2 to Banner Workflow 8.0” on page 36 for more information.

If you have any questions concerning your upgrade, you may call the ActionLine at 800-522-4827.

**Note**

When referencing scripts and directories from within this guide, a "/" was used. For MS Windows installations, the "/" should be replaced with a "\".
Hardware and Software Requirements

Banner Workflow - Client

Banner Workflow is a web based application. The requirements to run the client side of Banner Workflow are based on the browser that the user selects.

Banner Workflow has been tested against and supports the following browsers on machines with at least 256 MB of RAM:

- Microsoft Internet Explorer 6.x and 7
- Safari 1.3.2 and 2.0

**Note**

Please note that this list is for Banner Workflow only. If you are integrating workflow with other SunGard Higher Education products, such as Banner or Luminis, please check those products for the browsers they support.

**Note**

Banner Workflow uses Java Web Start technology to run client programs to support the graphical modeling of workflow and client launching from the browser. The first time a user attempts to use this functionality, they may be prompted to install a Java JRE if they do not already have a compatible one installed.

Banner Workflow - Server

**Required Software:**

- Oracle Application Server 10g Release 3 (10.1.3.4)
- Oracle Database 9i (9.2.0.4), 10g R1 (10.1.1) patch 2, or 10g R2 (10.2.0.1.0)
- JDK version 5 (with the Java Cryptography Extension)

The following versions of Banner are supported for integration

- Banner 7.x and Banner 8.x

The following versions of Luminis are supported for integration of single-sign on and Banner Workflow Channels

- Luminis 4.1.1 with the latest hot fix available from Luminis Support.
Note
Java may have been installed with your version of the Oracle Application Server or as part of your operating system. Check the version by executing `java -version` and install Java 5 separately if necessary.

Note
If your Java installation does not include the Java Cryptography Extension, you will need to install it separately before beginning the workflow installation. See “Setting Up Java” on page 18 for further details on how to download and install the JCE.

Required Hardware:

- Banner Workflow has been tested against a Sun Java 5 VM. All operating systems that are supported by the Oracle Application Server 10g Release 3 (10.1.3.4) should be acceptable for Banner Workflow. A majority of SunGard Higher Education’s testing has been done on a Windows 2003 Server.
- Memory: 2 GB of RAM.
- Hard drive: 1-2 GB of available space.
- Processor:
  - Intel: Dual 2.4 ghz processors are recommended.
  - Sparc: Dual 900 mhz processors are recommended.

Note
Operating systems such as HP-UX and Tru-64 will have their own Java VM. The VM must be built off of Sun’s Java 5 specification to run Banner Workflow.

Note
If you are running multiple engine instances, or are using clustering, it is CRITICAL that the system clock on each server be accurate. If the system clocks on the servers are out of sync, various pieces of workflow, such as SSO, and execution of automated activities may function erraticly, or generate unexpected errors. You should periodically check that the time on the servers is correct, or use a script or some other automated process to periodically check and update it.
Installing Oracle Application Server: Oracle Containers for J2EE

Banner Workflow requires the Oracle Application Server to be installed prior to installing Banner Workflow 8.0.

Note
To ensure that the scripts generated during the installation work properly, please do not select a java home directory that contains spaces.

Setting Up Java

Banner Workflow uses encryption to protect sensitive configuration information that is stored in the database. It does this by using the Java Cryptography Extension, which provides a framework for encryption which accepts pluggable providers. Banner Workflow will use the first available provider that supports the required encryption methods.

Note
Prior to Workflow 4.3, Workflow was hard coded to use Sun's reference JCE implementation, which required it to be manually installed on some platforms, such as AIX. Starting with the 4.3 release, Workflow can use whatever provider(s) the JVM is configured to use. See http://java.sun.com/products/jce/ for further details, as well as compatible 3rd party encryption providers.

If you install another JCE provider, your should install it as an extension.

CD Directory Structure

<table>
<thead>
<tr>
<th>&lt;dir&gt;</th>
<th>docs</th>
<th>Contains all of the documentation for Banner Workflow 8.0.</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;dir&gt;</td>
<td>support</td>
<td>Contains all support files for Banner Workflow 8.0 including the Banner examples.</td>
</tr>
<tr>
<td>&lt;file&gt;</td>
<td>installer.jar</td>
<td>Compressed file containing Banner Workflow 8.0.</td>
</tr>
</tbody>
</table>
Installation CD

Directory Structure

**WORKFLOW_HOME**

<table>
<thead>
<tr>
<th>Directory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>apache-ant-1.6.5</td>
<td>From <a href="http://ant.apache.org">http://ant.apache.org</a>. Ant is used in the installers and system utilities to provide platform-independent tools.</td>
</tr>
<tr>
<td>bin</td>
<td>This directory contains all of the scripts needed to maintain a Banner Workflow system.</td>
</tr>
<tr>
<td><strong>Note</strong></td>
<td>This directory will initially be empty during the installation process. A step within the installation will generate platform-specific scripts.</td>
</tr>
<tr>
<td>bootstraps</td>
<td>Contains the seed data necessary to build a Banner Workflow system.</td>
</tr>
<tr>
<td>css</td>
<td>Contains files that control Banner Workflow’s look and feel.</td>
</tr>
<tr>
<td>classes</td>
<td>Contains custom classes that can be used to alter the behavior of client-side scripts.</td>
</tr>
<tr>
<td>config</td>
<td>Contains all configuration information needed for Banner Workflow.</td>
</tr>
<tr>
<td>db</td>
<td>Contains all SQL scripts that are needed to create a Banner Workflow database and all Banner Integration tables.</td>
</tr>
<tr>
<td>engine</td>
<td>Contains files for running a default instance of the Banner Workflow Engine.</td>
</tr>
<tr>
<td><strong>Note</strong></td>
<td>This directory will be created during the installation.</td>
</tr>
<tr>
<td>examples</td>
<td>This directory contains examples of custom workflow components, workflows and tutorials for creating custom components.</td>
</tr>
<tr>
<td>lib</td>
<td>Contains the supporting workflow jars.</td>
</tr>
</tbody>
</table>
Create and Initialize Installation Directory

Create a directory that will be used for the Banner Workflow 8.0 installation. This directory will be referred to as WORKFLOW_HOME in all steps below. Consider naming the directory based on the type of Banner Workflow instance you are installing. For example:

If you are installing a TEST Banner Workflow environment, select TEST as the name of your WORKFLOW_HOME directory.

You could create the following directory structure:

/workflow80/PPRD
/workflow80/DEV
/workflow80/TEST
/workflow80/PROD

From the installation CD, copy installer.jar to the newly created WORKFLOW_HOME directory. Navigate to WORKFLOW_HOME and execute the following commands to unjar the installer.jar:

Windows: %JAVA_HOME%\bin\jar xf installer.jar
Solaris: $JAVA_HOME/bin/jar xf installer.jar
chmod +x ant
Ensure that the ORACLE_HOME environment variable is set to the Oracle Application Server home and run the ant script from WORKFLOW_HOME with no option:

Windows: ant
Solaris: ./ant

Note
Ant is the shell used for creating the initial deployment scripts and is bundled as a part of Banner Workflow.

Note
If multiple Banner Workflow instances are going to be installed please create a separate WORKFLOW_HOME for each instance.

Once you have created the WORKFLOW_HOME for the 8.0 installation, you can setup a new Banner Workflow system or migrate an existing 4.4.2 Workflow system. To install a new system, follow the instructions in “Installation” on page 21. To migrate an existing 4.4.2 instance to 8.0, follow the instructions in “Migrating from Workflow 4.4.2 to Banner Workflow 8.0” on page 36.

Installation

The following steps provide the necessary information to setup and install Banner Workflow for the first time. If you will be installing multiple instances of Banner Workflow, each Banner Workflow instance will require a separate WORKFLOW_HOME. However, more than one Banner Workflow instance can use a single Oracle Application Server instance. Specific instructions concerning best practices for the installation of multiple instances of Banner Workflow can be found in notes in the following steps.

Note
You can only have one WORKFLOW_HOME for each workflow instance. If you want to use clustering, you must use Oracle's clustering features, but you will still only have one WORKFLOW_HOME. Never install more than one WORKFLOW_HOME configured to use the same database account/schema as this may lead to data corruption.

Step 1  Create Oracle Connections

SunGard Higher Education recommends that a separate Oracle table space be created for Banner Workflow 8.0 named “WORKFLOW” with a default size of 500M.

Note
Each Banner Workflow instance will require unique WORKFLOW schemas that should be installed onto a tablespace separate from other applications. Multiple instances of Banner Workflow can share the same tablespace, but cannot share the same schemas.
An example script for creating tablespaces is provided, but needs to be modified depending on your file system. The script is located at:

```
WORKFLOW_HOME/db/workflow/users/
workflow_tablespace_creation_example.sql
```

Banner Workflow requires Oracle database connections to persist its data.

1. Create a user account to become the owner of the Banner Workflow tables. All tables in this workflow connection account are used solely for Banner Workflow and do not have to grant public access. The default tablespace for this account is “WORKFLOW”.

An example script is provided to create this user with the recommended name of “WORKFLOW” that is granted all the necessary rights. You will need to modify the script to specify the password you want to use. The script to create this user is located at:

```
WORKFLOW_HOME/db/workflow/users/workflow_user_creation.sql
```

2. To provide integration with Banner, Banner Workflow requires another user account to be setup with access to Banner. The tables owned by this Banner connection account are used to transfer data between Banner and Banner Workflow. The default tablespace for this account is “DEVELOPMENT” if the Banner recommended tablespace was used. The default tablespace should be the same default tablespace for all of the Banner users who will be launching Banner from Banner Workflow.

An example script is provided to create this user with the recommended name of “WFBANNER”. You will need to modify the script to specify the password you want to use. The script to create this user is located at:

```
WORKFLOW_HOME/db/banner/users/wfdcwfba.sql
```

When creating the “WFBANNER” user, a grant from BANINST1 for the GOKBSSF package is required for decrypting single sign-on credentials. To grant this permission, log in as BANINST1 and enter the following:

```
grant execute on baninst1.gokbssf to wfbanner;
```

This provides a WFBANNER account with the minimum grants needed to launch Banner from Banner Workflow. You will need to add additional grants to this account if you wish to use it for event handling or for use in automated activities.
3. To provide single sign on with Banner, Banner Workflow requires a limited user account to be setup within the Banner database. This user is granted access during the installation to call WFCKGSSO.GET_SSO( ) with encryption keys to allow transparent login to Banner from Banner Workflow. The default tablespace for this account is “DEVELOPMENT” if the Banner recommended tablespace was used.

An example script is provided to create this user with the HIGHLY recommended name of “WFSSO”. You will need to modify the script to specify the password you want to use. The script to create this user is located at:

```
WORKFLOW_HOME/db/banner/users/wfdcwsso.sql
```

As of Banner 7.2, some additional grants are necessary for the single sign on functionality to work correctly. Connect as system and issue the following grants:

- grant execute on WFCKGSSO to BANINST1;
- grant execute on WFIKWIBC to BANINST1;

**Note**
The sample script to create the wfsso user should not be run until after installing the workflow schemas for the first time in Step 3, below.

**Note**
Once Banner Workflow is running, you will need to log into Banner Workflow and modify the launch parameters of the Banner Forms Technology Type so the password for WFSSO matches the one specified in the database.

4. By default, the WFBANNER account in a new installation cannot be used to poll events from Banner. You must either create a Workflow Event user in the Banner database that has access to the Banner event tables as well as related sequences or grant the necessary privileges to the WFBANNER user.

Banner ships with installation scripts that create a WFEVENT account that can be used to poll for Banner events. Banner Workflow also includes an example script that can be used to create event polling accounts (or that you can use as a reference for the grants needed for the WFBANNER account if you want to enable it to poll events).

You will need to modify the script to specify the password you want to use. The script to create this user is located at:

```
WORKFLOW_HOME/db/banner/users/wfdcwfev.sql
```
5. By default, the WFBANNER account in a new installation does not have sufficient grants to serve as the account in the Banner Automated Data Source. This data source is used for processing both automated SQL and stored procedures. For setting up the Banner Automated Data Source, you may either extend the WFBANNER account by adding additional grants or create a separate database account such as WFAUTO (recommended). You may use the wfdcwfba.sql as a template for creating a separate account. It is recommended that you work with your Banner database administrator to grant access to Banner resources on an as-needed basis rather than simply granting 'DBA' to these accounts in a production environment.

**Note**

It is recommended that the WFAUTO database account is used with the Banner Automated Data Source.

**Note**

You must grant at least SELECT on the SPRIDEN table for the Banner Automated Data Source account to successfully complete the System Verification Workflow.

**Step 2 Update deployment information in configuration.xml**

Banner Workflow is a J2EE based application. The J2EE platform uses the Enterprise Archive (EAR) file as the format for software distribution. This specification defines the application-packaging conventions that are necessary for using EAR files as self-describing Field-Replaceable Units.

The configuration.xml file, located in the config directory off of WORKFLOW_HOME, stores the information needed to deploy a Banner Workflow instance and control its runtime behavior. This information is used to build an EAR file, update data source connections, and configure ports for Banner Workflow.

In general, changes made to the <Deployment> section of the configuration file (with the exception of changing logging settings) require the application to be redeployed to take effect. All other changes require the new configuration to be uploaded to the database and the application to be restarted. For complete details on the changes that require a redeploy, see “configuration.xml” on page 100.

There are two ways to update configuration.xml prior to deployment.

- Manually edit the file.
- Use the ConfigWizard, which walks you through the most common changes needed.
Manually Edit configuration.xml

If you choose to edit the configuration file directly, you should consult the documentation on the file's structure in “configuration.xml” on page 100, then update the following sections:

1. Under <Deployment> update <WorkflowDataSource> and <BannerDataSource> to use the database accounts you set up in “Create Oracle Connections” on page 21.

2. Update the <ApplicationName> to be a name unique for this workflow installation, such as wfProd.

3. Update the <ApplicationServerHost> to the hostname of the server.

>Note
   If using clustering, set it to the hostname of one of the servers in the cluster.

4. Update the <WebApplication> element to specify the protocol, host, port, and root for the web application. If you are using clustering, the WebApplication should specify the “front end” that user will logon to. For example, if you are using a load balancer, that address should be specified here.

5. Under the <Engines> element, update the <EngineConfiguration name="main"> element to configure the rmi port the engine should run on, as well as the tuning parameters. If you intend to run a single engine instance, and are not using engine clustering, the default installation will install a single engine instance using the “main” engine configuration. You should edit this “main” engine configuration and adjust the rmi port and thread pool settings as desired.

   If you are using engine clustering, or if you intend to deploy multiple engine nodes that require different configuration options (such as a different port), add additional <EngineInstance> and <EngineConfiguration> sections as necessary. Be sure to give each <EngineConfiguration> section a unique name. For each engine instance that you plan to install, add an <EngineInstance> element to the configuration file, specifying the hostname and configuration the engine will run under. See “Deploying and Managing Multiple Engine Instances” on page 43 for details on managing multiple engine instances.

6. Update the <EmailServer> settings if you want to use email notifications.

7. Update the BannerDatabase datasource under <DataSources> to use SQLStoredProcedures from Banner Workflow.

8. Update the Banner W-Event Provider #1 event provider if you want workflow to poll for Banner events.
Configuration Wizard

If you want to use the wizard to perform all of the above steps, run “bin/configwizard”. You will be presented with a menu of options. For the initial install, you should select the option to run all the wizards. The following sections describe the values that you should provide to each wizard as it runs. At each prompt, the current value set in the configuration file will be provided in square brackets "[ ]". To leave any value unchanged, simply press Enter.

Workflow Data Source wizard

This wizard updates the <Deployment>/<WorkflowDataSource> element with the JDBC connection and connection pool properties. You will be prompted to provide the following information:

**URL for the Workflow Connection**

This is the connect string for the workflow database account you setup in “Create Oracle Connections” on page 21 and should be formatted as jdbc:oracle:thin:@<servername>:<listener port>:<sid or service>. For example:

```
jdbc:oracle:thin:@myserver:1521:workflow
```

**Username for the Workflow Connection**

This is the username for the workflow account you setup in “Create Oracle Connections” on page 21.

**Password for the Workflow Connection**

This is the password you setup for the workflow account in “Create Oracle Connections” on page 21.

**Maximum Number of Connections for the Workflow Pool**

This is the maximum number of connections to the workflow database that the application server may use at one time, and affects the total number of users that can make requests of the server at once.
Banner Data Source wizard

This wizard updates the `<Deployment>/<BannerDataSource>` element with the JDBC connection and connection pool properties. You will be prompted to provide the following information:

**URL for the Banner Connection**

This is the connect string for the workflow database account you setup in “Create Oracle Connections” on page 21 and should be formatted as `jdbc:oracle:thin:@<servername>:<listener port>:<sid or service>`. For example:

```
jdbc:oracle:thin:@myserver:1521:workflow
```

**Username for the Banner Connection**

This is the username for the “WFBANNER” account you setup in “Create Oracle Connections” on page 21.

**Password for the Banner Connection**

This is the password for the “WFBANNER” account you setup in “Create Oracle Connections” on page 21.

**Maximum Number of Connections for the Banner Pool**

This is the maximum number of connections to the Banner database that the application server may use at one time, and affects the total number of users that can make requests of the server at once.

Application Server wizard

This wizard updates the `<ApplicationName>` and `<ApplicationServerHost>` elements under `<Deployment>`. You will be prompted to provide the following information:

**Application Name**

The name chosen here is the name that is used within the Oracle Application Server to identify the deployment of workflow, and is also used to form the name of the ear file. The default name is “workflow”. Each Banner Workflow instance should have a unique application name. The application name should follow the standard of incorporating the application type into the name. For example, a system in workflow80/PROD should be given a name like “wfprod”.

**Application Server Host**

This is the hostname of the server that the OC4J instance that will run workflow is on.
Web Settings

This wizard updates the <WebApplication> element under <Deployment> to specify the protocol, host, port, and root under which the web application portion of Banner Workflow will run. You will be prompted to provide the following information:

Protocol

This should be either http, or, if you are running under ssl, https.

Host

This should be set to the hostname of the server Banner Workflow is deployed to. It will be used by other workflow components (such as Luminis) that need to contact Banner Workflow.

Port

Should be set to the port that the Oracle Application Server Apache component is running on (default is 7777 for http).

Root

This is the webroot of the workflow application, by default "/workflow". This should be set to a unique value for each Banner Workflow instance running on the server, and should be given a name that identifies the type of installation, for example "/wfprod" for a production system.

Note

The <WebApplication> element does not control the HTTP server settings. HTTP server settings must be managed individually, depending on your HTTP server configuration or clustered environment. The <WebApplication> element tells Banner Workflow how your HTTP server is configured, so Banner Workflow can configure items such as Banner Workflow channels, the import/export utility, the Modeler, and how to contact Banner Workflow through the web server.
Workflow Engine

This wizard sets up engine instances and manages configurations for the instances through a series of submenus. You can add and remove engine instances, edit, remove, and copy engine configurations, and change the engine administrative password.

Adding and Removing Engine Instances

An engine instance in the configuration tells the application server that it can expect an instance of the workflow engine to be running on the specified host that will be running under the specified configuration. Using this information, the application server can forward requests to the engine.

Each engine instance that is started must run under an engine configuration. The engine configuration tells the engine what port to bind to, as well as how to size its thread pools in order to perform work. Engine instances that are running on different hosts can run under the same configuration. In general, you should run all engine instances under the same configuration, unless you need an instance to run on a different rmi port, or want to change the number of threads available to its pools.

If you intend to run a single engine instance, edit the "main" engine configuration and set the rmi port and thread pool options as desired. If you intend to run multiple engine instances, you will need to edit and add additional engine instances and engine configurations to support your installation. See “Deploying and Managing Multiple Engine Instances” on page 43 for details on configuring and deploying multiple workflow engines.

Note

To add a new engine configuration, you must first select an existing one to edit, then choose to copy it to another name. You can then edit the newly copied configuration.

When you choose to edit an engine configuration, you will be prompted to provide values for the following settings:

Engine RMI Port

This is the RMI port that the engine will run on. It must be a port that no other applications on the server will use.

Maximum Engine JDBC Connections

This controls the maximum number of JDBC connections the engine's pool can have for servicing incoming requests.
Main Pool Max Threads

This sets the maximum number of concurrent threads the engine can use to perform asynchronous work such as starting workitems, routing items to worklists, etc. See the <Engine> section of the configuration documentation and “Tuning the Workflow Engine” on page 165 for details.

Main Pool Min Threads

This sets the minimum number of threads the engine will keep active for performing normal asynchronous work.

Notification Pool Max Threads

This sets the maximum number of concurrent threads the engine can use to perform notification work, such as email notification of item availability or lagging/overdue notifications. If you do not use a large number of email notifications that work items are ready, are not using Workflow Metrics, or do not have a large number of lagging/overdue notices, this number can be kept small.

Note

If you notice that the ENG_EVENTS table frequently has more than ten rows with scheduled_time NULL and event_type that starts with "immediate.notification", then you should increase this value.

Notification Pool Min Threads

This sets the minimum number of threads the engine will keep active for performing notifications.

External Events Max Threads

This sets the maximum number of threads to use to evaluate external events. See “External Events” on page 139 for further details on External Events.

External Events Min Threads

This sets the minimum number of threads to use to evaluate external events.

You can also choose to change the engine administrative password. This will set the administrative password for the engine(s). Client-side utilities that can direct the engine to shutdown must provide the engine with this password. You can also choose to change the default maximum memory settings for the VM that the engine will run in.
Email

This wizard allows you to set the server to use for sending email. You will be prompted to provide the following information:

Email Server

This value is the address of the smtp server that Banner Workflow should use to send emails.

DefaultAccount Address

The default email account will be used for the Banner Workflow environment to send out system notifications.

Banner Automated Activity Data Source

This wizard will give you the option to copy the Banner database JDBC Url from the <BannerDataSource> element under <Deployment> to the default stored procedure datasource in the <DataSources> section of the configuration. You will be prompted to provide the following information:

Copy

Select yes to copy the information, or no to leave the settings as they currently are.

Username

Specify the username to use when connecting to the database for running sql stored procedures.

Password

Specify the password to use when connecting to the database for running sql stored procedures.

Banner Event Dispatcher

This wizard will give you the option to copy the Banner database JDBC Url from the <BannerDataSource> element under <Deployment> to the default event provider under the <EventDispatcher> section. You will be prompted to provide the following information:

Copy

Select yes to copy the information, or no to leave the settings as they currently are.
Username

Specify the username to use when connecting to the database for event polling.

Password

Specify the password to use when connecting to the database for event polling.

Root Account Password

This wizard allows you to change the passwords for the root account (wfroot) and the web services account used for bootstrapping the system, server-to-server web services communication, and administration scripts. You will be prompted to provide the following information:

Password for Root (wfroot)

Set the password.

Password for Web Service Account (wfwebservice)

Set the password.

When Finished

After completing the wizard, select "Save" to apply your changes to the configuration file and then "Quit" to exit.

Step 3 Prepare to Deploy

With the database(s) used by Banner Workflow and Banner running, execute the following:

```
bin/wftool all
```

This will apply the changes made to the configuration, install the schemas to the database (you will be prompted to confirm changes that will affect the database), install a default engine node, and create the ear file for deployment.
Step 4 Deploy the EAR file

The ear file created by the installer must be deployed to an OC4J instance in the Oracle Application Server. Banner Workflow can be deployed to Oracle Application Server 10.1.3.4.

1. The ear file should be deployed to a new OC4J instance that has no other application deployed to it. This is typically done using the createinstance script found in Oracle AS bin directory but check the Oracle documentation for specific instructions for creating an oc4j instance. For example:

```
[ORACLE AS HOME]\bin createinstance -instanceName wf80prod -groupName wf80prod
```

2. Open the Server Properties view of the selected OC4J instance within Enterprise Manager.

3. Set the heap memory sizes. Typically, 1024M for the maximum heap size and 512M for the initial heap size.

4. You must also add the following option to the Start-parameters: Java Options of the Server Properties for the OC4J instance:

```
-Dlog4j.customConfigurationPolicyClass=com.sungardhe.workflow.util.logging.ConfigurationLoggingPolicy
```

5. Under the Start-parameters:OC4J options, add the following two options (if not already present):

```
-userThreads
-properties
```

6. Deploy the ear file by selecting WORKFLOW_HOME/dist/workflow.ear for deployment, and select Automatically create a new deployment plan for the Deployment Plan.

7. On the Deploy: Application Attributes screen, you should see the Application Name set to the same value as the Application Name value in the configuration.xml. Follow the prompts to get to the next step and complete the deployment.
Step 5  Start the Workflow Engine

To bring up a Banner Workflow system, you must start at least one Workflow Engine. The Workflow Engine is the component that drives the running processes. Each deployed engine instance has a start script. By default, an engine instance using the 'main' engine configuration is deployed to the WORKFLOW_HOME/engine directory during installation. To start this default Workflow Engine, execute:

```
WORKFLOW_HOME/engine/bin/startengine
```

See “Deploying and Managing Multiple Engine Instances” on page 43 for more information.

Step 6  Start the system

Once the ear file is deployed, start the system by starting the oc4j instance.

Step 7  Import base data

Banner Workflow requires base data that needs to be imported prior to use. The import command will import the necessary data. OC4J must be running prior to starting an import.

To import base data, execute the following command:

```
WORKFLOW_HOME/bin/import wfroot <password for wfroot> WORKFLOW_HOME/
bootstraps/basedata.zip
```

The base data will contain an admin and analyst account to start with. The username for the admin account is “admin” with a default password of “u_pick_it”. The username for the analyst user is "analyst" with a default password of "u_pick_it".

⚠️ Note

The password for wfroot is the password you set for the root accounts if using the ConfigWizard, or the password you set on the <SuperUser> element under <SecurityIntegration> in the configuration.xml. By default, it is "password".
Re-deploying after configuration changes

If you make changes to the configuration file that require rebuilding/redeploying the ear file, or the workflow.car, or the engine node(s), you can apply all the changes and rebuild all the artifacts by running:

```bash
bin/wftool updateSystem
```

This command will rebuild all the scripts, upload the configuration changes to the database, rebuild the car file (if Luminis integration is enabled) and rebuild the default engine node and engine installers. (Note: you only need to physically re-install engine nodes if you have changed the url, username, or password of the workflow data source. Any other engine changes will be downloaded from the database when the engine node(s) are restarted.)

After updating the system, redeploy the ear.
Migrating from Workflow 4.4.2 to Banner Workflow 8.0

Before beginning a migration from 4.4.2, review “Hardware and Software Requirements” on page 16 and follow the instructions in “Create and Initialize Installation Directory” on page 20 to create the WORKFLOW_HOME for the 8.0 installation.

Note
Workflow 8.0 is only certified against 10g Release 3 (10.1.3.4). While your existing 4.4.2 system can be running on an older version of Oracle Application Server, once you complete the migration steps, you must deploy Workflow 8.0 into a supported version of OAS.

Warning
Prior to performing a migration from Workflow 4.4.2 to Banner Workflow 8.0, it is highly recommended that complete database backups of the WORKFLOW schema be created after shutting down the old workflow system. Without this backup, it will be impossible to recover the 4.4.2 system if anything goes wrong during migration.

Warning
Banner Workflow 8.0 transfers import/export data over either http/https. If your <WebApplication> element in your configuration.xml uses https as the protocol, your Java installation must trust the SSL certificate being used by your web server. This typically requires action on your part if you are using self-signed certificates. Otherwise, Step 4 of the Migration will fail. See Appendix, “Web Service Applications” for details on how to import self-signed certificates into your java installation's keystore.

Note
In this section, PREVIOUS_HOME refers to the directory of your Workflow 4.4.2 installation and WORKFLOW_HOME refers to your Banner Workflow 8.0 directory.

The migration process will use the same datasources, database accounts, and schemas for the new 8.0 system that are being used for the 4.4.2 system. During migration, the data in the 4.4.2 system will be converted (as necessary) to be compatible with 8.0. Once the conversion process begins, you will no longer be able to start and run the old 4.4.2 installation. After the migration is complete, all workflows and data present in the 4.4.2 system will be available in the 8.0 system. Once migration begins, you should not use any of the Workflow 4.4.2 components except where explicitly directed to by the migration process.

Do not make any changes to the workflow configuration during migration. For example, do not change datasources or any other settings. If you want to make configuration changes, either make them to the 4.4.2 system and deploy/test them there before beginning migration, or complete the entire migration, then make the changes to the 8.0 system and redeploy (see “Re-deploying after configuration changes” on page 35).
The migration will proceed in a series of steps. Unless otherwise indicated, if any step fails, you must stop and correct the issue before continuing. Failure to do so will prevent a clean migration, and may result in a loss of data.

Detailed output on each migration step, including errors encountered will be logged to WORKFLOW_HOME/migration/migration.log.

The migration process attempts to strike a balance between performance and database size. Where necessary, data from the 4.4.2 system is converted in a batch. However, the conversion process may temporarily use more tablespace than the final system will need once conversion is complete. If the tablespaces used for the 4.4.2 system are near capacity, it is strongly recommended that you increase their size before beginning migration.

Note
All migration steps must be executed from the 8.0 WORKFLOW_HOME. For more information on setting up your WORKFLOW_HOME directory, see “Create and Initialize Installation Directory” on page 20.

Step 1  Export Previous Data

The first step is to export "bootstrap" data with primary keys from the 4.4.2 system. First start the Workflow 4.4.2 system (if it isn't running) and go to your WORKFLOW_HOME and execute the following task:

```
migration/bin/migrate step1
```

The script will prompt you for the location of the previous (4.4.2) home. Any further steps that need that location will remember it, but will give you the chance to change it if you entered it incorrectly.

This step may take some time, depending on the number of components and workflows in your system.

After Step 1 completes successfully, you should stop the 4.4.2 system, and backup the 4.4.2 Workflow schema. If any further steps fail, you can restore the database tables, and restart your migration from Step 2.

Note
After Step 1, please verify that the export of the previous data (WORKFLOW_HOME/migration/PreviousDataForMigration.zip) is not a zero-length file (0 bytes).
Step 2  Migrate Configuration and Backup Tables

This step will migrate the previous Workflow configuration to the 8.0 system, and prepare the Workflow schema for later migration steps.

⚠️ Warning
Before you run Step 2, you must undeploy the Workflow application from the OC4J instance and then stop that OC4J instance. If you do not undeploy your application and stop the OC4J instance, prior to running Step 2, you run the risk of destroying your runtime data.

Execute the migration by running:

```
migration/bin/migrate step2
```

⚠️ Note
This step is designed to only be performed once during a migration. Running it again will erase runtime data, making it impossible to safely complete migration without manually restoring from the backup you made after Step 1.

Step 3  Prepare for Deployment and Deploy

This step will perform most of the steps involved in a normal installation, including initializing the Workflow and Banner integration schemas, and building the workflow ear, default engine node, engine installer, and workflow car file (if Luminis integration is enabled). To execute this step, run:

```
migration/bin/migrate step3
```

Deploy the ear file (and extra engine instances if you are using clustering) using the instructions in “Deploy the EAR file” on page 33. Be sure that the application is deployed with the correct Application Name and Web Application name as specified in the configuration.xml using the same name/port settings as the previous 4.4.2 system.

⚠️ Note
The log4j property in Server Properties has changed in Banner Workflow 8.0. To make sure that the new log4j property value is applied, please see “Deploy the EAR file” on page 33 for details.

Step 4  Import Previous Data

This step requires that the OC4J instance for the new 8.0 system be running. Start the OC4J instance on your application server using Enterprise Manager.

This will import the previous 'administrative' data, such as user accounts, roles, components, workflow definitions, etc. into the new system.

To execute this step, run:

```
migration/bin/migrate step4
```
If the step encounters errors, details of the import may be found in WORKFLOW_HOME/migration/migrationimport.log

⚠️ Warning
Any errors encountered in this step are very serious. Any workflow definitions that fail to import will prevent all instances of started workflows (completed or running) present in the previous installation from migrating to the new system or may prevent migration from completing at all.

⚠️ Note
If you encounter the following warning, you may ignore it:

```
WARN org.apache.commons.httpclient.HttpMethodBase - Going to buffer response body of large or unknown size. Using getResponseBodyAsStream instead is recommended.
```

### Step 5  Restore Constraints

Before performing this step, shutdown the workflow system. To stop the OC4J instance for your instance of Workflow, use the Enterprise Manager.

Do not attempt to start an engine instance. During migration, the engines are automatically disabled, and attempting to start them will cause an exception.

```
migration/bin/migrate step5
```

### Step 6  Re-enable Engines

During earlier migration steps, the engines and automated activity handlers were disabled to prevent them from processing data during migration. This step will re-enable these services.

```
migration/bin/migrate step6
```

### Step 7  Cleanup

Step 7 will drop all intermediate tables used during migration. It is not possible to replay any of the migration steps once step 7 is run.

```
migration/bin/migrate step7
```

### Step 8  Update Clustered Engines and Deploy CAR

If you are using Luminis integration, you should re-deploy the CAR file at this point. See “Integrating Banner Workflow with Luminis” on page 75 for more information.

⚠️ Note
If you are using Workflow's Luminis channels, your channel definitions will need to be updated because the channel class names have changed in Banner Workflow 8.0. Please see “Channel Administration” on page 80 for the correct channel class names.
If you are running multiple engine nodes, or have installed an engine node to a non-default location, you must now reinstall each node. See “Deploying and Managing Multiple Engine Instances” on page 43 for more information.

At this point the 8.0 system has been deployed and contains the data from the previous 4.4.2 system.

### Starting and Stopping Banner Workflow

#### Starting Banner Workflow

1. Start at least one Workflow Engine. To start the default engine, execute `WORKFLOW_HOME/engine/bin/startengine`. See “Deploying and Managing Multiple Engine Instances” on page 43 for details on managing multiple engines.

2. Start the Oracle Application Server.

3. Ensure that the OC4J instance within the Oracle Application Server is started. How you do this depends on which version of the application server you are using (dcmctl for Release 2, opmnctl for Release 3, or the Enterprise Manager). See your Oracle documentation for further details.

4. Launch a supported web browser and navigate to the Banner Workflow web site you created. The URL for this website will be determined by the `<WebApplication>` element that you set up in the configuration.xml file. For example:

   http://school.edu:7777/workflow

**Tip:**

Starting the Workflow Engine requires an active process window. On Windows Servers, you can launch the process in a new window by adding "start" to the call to startengine. For example:

   start engine\bin\startengine

This will create a new command window that will run the Workflow Engine.

On a Unix based machine, the startengine script uses "nohup" to run the Workflow Engine in a background process and directs the output to `WORKFLOW_HOME/engine/bin/startengine.out`. 

Stopping Banner Workflow

1. Either via the Oracle Application Server Enterprise Manager or command line tools (dcmctl for Release 2, opmnctl for Release 3, see Oracle documentation for details).

2. Stop the Workflow Engines. To stop all the Workflow Engines execute the following:

   \`
   WORKFLOW_HOME/engine/bin/engineconsole stop -all -password <engine password>
   \`

   The engine password is defined in the configuration.xml file, and defaults to "password".

Supported Integration Modes for Banner Workflow

Banner Workflow supports the following authentication modes for integration with Luminis and Banner with the following considerations:

- Workflow Authentication Mode - Integration with Luminis is not supported in this authentication mode. For more information on Banner authentication see “Banner Integration” on page 47.

- Workflow External - For more information on Banner authentication see “Banner Integration” on page 47. For more information on Luminis authentication see “Configuring Luminis for Workflow External Authentication” on page 76.

- CAS - For more information on Banner authentication in CAS mode, see “Banner Integration” on page 47. For more information on Luminis authentication in CAS mode, see “Configuring Luminis for IDM or CAS Authentication” on page 79.

- IDM Gateway - For more information on Banner authentication using the IDM Gateway, see “Banner Integration” on page 47. For more information on Luminis authentication using the IDM Gateway, see “Configuring Luminis for IDM or CAS Authentication” on page 79.

Setting up Banner Integration

Please refer to Chapter 2 “Banner Integration” on page 47 for information on how to configure your Banner Technology Types, Event Provider, and Data Sources.
Setting up Luminis Integration

Please refer to “Integrating Banner Workflow with Luminis” on page 75 for information on how to configure your Luminis and Banner Workflow environments.

Configuring Banner Workflow without Banner

A typical Banner Workflow install generally involves a data source connection to a Banner database. If you will not be integrating with Banner, please note the following:

**Note**

Some pieces of Banner Workflow expect Banner integration tables to be in place. Use the same values for the Banner data source that you use for the Workflow data source, except for the max connections property. For Maximum Number of Connections for the Banner Pool, specify 1. See “Update deployment information in configuration.xml” on page 24 for more information about these settings.

After making the changes listed above starting the installation from step 1, essentially what occurs when going through the installation is that the single sign on tables which are typically created in a Banner tablespace will now be created within Banner Workflow. The tables will never be used, but need to be there for some of the internal services within Banner Workflow to perform properly.

Running the Workflow Engine as a Windows Service

On a Windows Server, the Workflow Engine can be run as a startup service or daemon by installing the Workflow Engine as a Windows Service. This allows a system administrator to bounce the server without having to manually authenticate back into the server and manually restart the Workflow Engine.

The service script can be found in the workflow engine bin directory, and can be used for installing, starting, and removing the workflow engine as a Windows Service. The workflow engine is usually installed as a subdirectory in the workflow installation home, but can be installed outside of it, especially when creating additional engine nodes.
Usage: service { start : stop : restart : install : remove }

- start - Starts the workflow engine. This is the same as going to the Services area under Windows Administrative Tools to start the service.
- stop - Stops the workflow engine.
- restart - Restarts the workflow engine.
- install - Installs the workflow engine as a service.
- remove - Removes the workflow engine as a service.

For example, to install and start the engine as a service:

```
{workflow engine directory}\bin\service install
{workflow engine directory}\bin\service start
```

**Log Files**

The workflow engine service writes log output to `{workflow engine directory}\bin\engineservice.log`.

**Deploying and Managing Multiple Engine Instances**

To increase scalability and provide failover support, you can run multiple instances of the Workflow Engine.

**Overview**

A Workflow Engine is a java program that runs in a regular java VM. An installed workflow system can make use of multiple engine instances (each running in its own VM) on one or more servers in order to provide scalability and transparent failover.

⚠️ **Warning**

Certain aspects of running multiple nodes (like transparent failover of work) require time-based checks of records. It is critical that the system clock on any server any engine node is deployed on is accurate. If multiple engine nodes are run on servers that have their system clocks out of sync, failover and workflow metrics may not work as well as expected, or may result in inaccuracies.

The engine instances that are installed are controlled by one or more 'named' engine configurations. Each engine configuration specifies, among other things, the port that the engine runs on, and the size of its thread pools.
For each engine instance that will be running, the `<Engines>` element in the configuration must contain an `<EngineInstance>` element listing that instance. The `<EngineInstance>` elements are used by each application server node to locate running engines that they can forward requests to. When a request that requires the use of an engine instance is received by the application server, it will automatically select a running engine from the list of known engines from the `<EngineInstance>` list to forward the request to.

If an engine instance is not running, the workflow system will automatically try to forward the request to another instance, until it either finds a running instance or exhausts the list. The workflow system will also periodically check instances from the `<EngineInstance>` list that are not running to see if they have been started. So the workflow system as a whole can automatically respond to individual engine instances being started or stopped. However, the check to see if an engine instance in the list has been restarted can cause occasional delays to user requests; therefore, if you permanently remove an engine instance, you should also remove its `<EngineInstance>` element in the configuration. Only engine instances that you intend to always have running (barring unanticipated failures or routine maintenance) should be listed in an `<EngineInstance>` element.

Each engine instance acquires the configuration from the database when the instance is started, so the configuration in the database must be up to date (by running 'bin/wftool uploadconfig' after any configuration changes) before starting an engine instance.

If you are using appserver clustering, you should ensure that each `<EngineInstance>` element refers to the hostname of the server running the engine, not just "localhost". This applies even if you are only running the default engine instance. (By default, the engine instance element for this is 'localhost', so you should change it to the actual name of the server WORKFLOW_HOME is on if you are using clustering.)

**Install an Engine Instance**

There are three aspects to configuring a multi-engine instance system:

- providing configuration information to each engine instance
- deploying each engine instance
- making the workflow system aware of each instance

Each engine instance that runs is controlled by an engine configuration. An engine configuration is a named section of the configuration that tells an engine what rmi port to run on, as well as how to size its thread pools. It is not necessary to have a separate named engine configuration for each engine instance; multiple instances can use the same configuration, as long as they are deployed to different hosts so that they can run on the same port. The only time you need to provide multiple configurations is if you need engine instances to run on different rmi ports, or want them to use different sized thread pools. You can edit, copy, and delete engine configurations using the configuration wizard, or by editing the configuration.xml file directly.
During workflow installation, a default engine node is installed to the engine directory under WORKFLOW_HOME. This default node will use the 'main' engine configuration. By default, the configuration contains an <EngineInstance> element pointing to this engine. If you do not intend to run this engine instance, delete the <EngineInstance> element pointing to it, and replace it with one pointing to your engine's location.

To deploy an engine instance to a separate location, follow these steps.

1. Choose the server that you will deploy the new engine instance to. The server must have a Java 5 VM installed. Decide which engine configuration this instance will use. Edit the configuration or add a new engine configuration if necessary.

2. Edit the configuration and add an EngineInstance for the new engine, specifying the server you have chosen to deploy to and the named engine configuration the instance will use.

3. Run 'bin/wftool uploadconfig' to store the configuration changes in the database. If you have not yet deployed a workflow system, run 'bin/wftool engine' to create the engine installer.

4. On the server you wish to deploy the new engine instance on, create a directory to hold the engine. Copy the engineinstaller.jar from WORKFLOW_HOME to this directory.

5. Extract the contents of engineinstaller.jar to your chosen engine directory with 'jar xf engineinstaller.jar'. (This assumes that java is in your path; you may need to specify the full path to the jar executable if java is not in your path, e.g., /usr/java/jar). You can delete the engineinstaller.jar after you have extracted its contents.

6. Finish the engine deployment by setting it up with the correct configuration. To do so, run 'java -jar engine.jar -install'. This will display the list of available engine configurations and prompt you to select the one this engine instance should use.

**Note**

The installer obtains this information from the configuration stored in the database, so the database must be running and must contain an up to date copy of the configuration for this step to work.

After you choose the configuration, the installer will create a bin directory and place a startengine and engineconsole script in it. The startengine script will start an instance of the engine using the configuration you chose. The engineconsole script will allow you to control this (and other engine instances). If you want to change the configuration this instance uses, you can run 'java -jar engine.jar -install' again to recreate the scripts.
Note

The install step can be run in a non-interactive mode by specifying the name of the configuration to use after the `-install` parameter. For example, '
java -jar engine.jar -install main' will setup the engine instance to use the 'main' engine configuration.

7. Finally, start the new engine instance, and restart the application server so it sees the configuration changes and can begin forwarding requests to the new engine instance.

Example

Assume we have completed a workflow installation on server "myhost1.myschool.edu" and are running the default engine instance that was installed under WORKFLOW_HOME on myhost1. Now we want to install and use a second engine instance on myhost2.myschool.edu. The second instance should be identical to the first (run on the same port, use the same size pools, etc.)

First, we need to make the workflow system aware of the second engine instance. Using WORKFLOW_HOME/bin/configwizard, add a new engine instance, specifying 'myhost2.myschool.edu" as the host, and 'main' as the configuration. (This tells the workflow system that it can expect a second workflow engine instance to be running on myhost2.myschool.edu, and that instance uses the 'main' configuration. This allows the workflow system to locate the instance and forward requests to it.)

Next, run "bin/wftool uploadconfig" to load the updated configuration to the database.

Create an engine directory on myhost2 and copy the engineinstaller.jar to it. Extract the contents with 'jar xf engineinstaller.jar' and complete the engine installation with 'java -jar engine.jar -install', selecting the 'main' engine configuration. Start the new engine instance with the bin/startengine script that was created, and restart the application server so it sees the changed configuration and can begin forwarding requests to the new engine instance.

Note

Once an engine node is installed, you can alter it's properties by changing the configuration.xml and uploading the changes to the database, then restart the engine node. The only time you need to re-generate the engine installer and unzip it to all engine directories is if you change the workflow datasource properties, i.e., if you change the database account workflow runs under. In this case, it is necessary to use 'bin/wftool updateSystem' to rebuild all artifacts, to re-create the installer with the (encrypted) database connection information, and re-install each engine node.
2 Banner Integration

Banner Workflow has the ability to integrate with many applications and is delivered ready to integrate with Banner. The following functionality was created to achieve integration with Banner:

- The ability to respond to events that occur in the Banner Database.
- The ability to open Banner forms interactively as a work item in the Banner Workflow system.
- The ability for bi-directional passing of data or parameters between Banner Workflow and Banner forms.
- The ability to use an automatic component to allow Banner Workflow to make stored procedure calls to the Banner database to retrieve data or perform work as part of a workflow.
- The ability to use SQL to query tables in the Banner database as part of a workflow and return values into workflow context.

To respond to events, an EventProvider for Banner is used by Banner Workflow. This provider knows how to pick up events and event parameters from Banner generated event tables and mark the event's disposition.

Banner forms can be launched from Banner Workflow as an interactive work item. This means that the user launches Banner from their worklist, passes workflow context parameters into the Banner form's KEY_BLOCK, performs some work, and then communicates to Banner Workflow that the work is complete and that the workflow should progress.

For workflows that interact with Banner, a way to automatically retrieve data or perform work against the Banner database to add relevant information to a workflow is necessary. For example, if Banner Workflow has picked up an event from Banner indicating that a new gift was received, you might want to use the donor's ID to retrieve the total amount they have given to the University before the Director of Development is notified. This way, the Director of Development can see additional information before thanking the donor.

There are two ways to gather this information, through an automatic SQL Query or through a SQL Stored Procedure against the database. Both can be set up to take in the donor's ID and retrieve the information via SQL syntax or by calling a PL/SQL procedure that has been created to get the information.
Banner Integration Overview

This section details the main components of Banner with Banner Workflow integration.

Event Service

Workflows have the capability to be started automatically by events. The Workflow Event Service polls for events from Event Providers, evaluates any guard conditions to determine which workflows to start, and then maps any parameters to the workflow.

Based on the settings in the configuration.xml file, the EventDispatcher requests new events from the BannerTableExternalEventProvider for the specified Target name in a Banner database. If the event that is picked up matches an event defined in the Banner Workflow system and all required parameters defined in workflow have been passed from the Banner event, the event is considered valid and will be evaluated to determine if it should start any workflows. See “Setup Event Processing in Banner Workflow” on page 54 for more information on setting up an event in Banner to be picked up by the BannerTableExternalEventProvider.

The EventEvaluator handles the evaluation by looking at all workflows that are associated with a particular event and checking any guard conditions to see if it should start those workflows. If the conditions pass, the evaluator will start the workflow(s) and map the event provider's parameters into the initial context parameters of the workflow.

Technologies to Launch Banner

When a user chooses to launch a Banner component from their worklist, Banner Workflow will determine if Banner has already been launched and is waiting at the GUAGMNU form to pick up new work. If it is not, Banner Workflow will launch a new Banner session in internet native mode.

During a new launch, Banner Workflow passes arguments to the Banner session to establish a link between the two sessions and to facilitate single sign on. The specific arguments are explained in greater detail in the section on the “Banner Launcher Service” on page 50.
Once Banner Workflow has logged the user onto Banner with the same user ID and password that they have set up in Banner Workflow, the Banner session will immediately check the work item queue for work and pick up the work item to start the task that was launched from the worklist.

**Note**

If a Banner session has already been launched from Banner Workflow and is waiting at the GUAGMNU form, it will already be checking the work item queue for the new task. This is why Banner Workflow may not need to launch an additional Banner session.

1. For work item manipulation from Banner, HTTP communication via UTL_HTTP is used to communicate requests to Banner Workflow and JDBC for Banner Workflow is used to communicate information back and forth with the Banner session. Using these technologies, the Banner session requests the form name to launch, and the parameters to use, when launching the form. The Banner session also uses the same technology to set any new parameters and to complete or release the work item.

2. To test that the HTTP communication from the Banner database to the Banner Workflow server is working properly:

   2.1. Log into SQL plus as the 'wfbanner' user described in the configuration.xml file. Ensure that you use the same database that is in the configuration.xml file.

   2.2. Execute the following SQL command:

   ```sql
   SELECT UTL_HTTP.REQUEST('http://workflow.school.edu:7777/workflow') FROM DUAL;
   ```

   where `http://workflow.school.edu:7777/workflow` is the name of the URL that you use to access your Banner Workflow environment.
2.3. If the command returns HTML back and comes back relatively quickly, then UTL_HTTP appears to be in place and the network connection between the Banner Database and Banner Workflow can be established.

If the command appears to hang or times out after a while, the URL that is being used is typically incorrect or the workflow server is currently inaccessible. At this point the issue is a network one and can be different for each installation. Please contact your network administrators for assistance.

Banner Launcher Service

When a Banner work item is launched, the Banner Launcher Service configuration determines how the launch is performed. The configuration of the Launcher Service is handled in the Banner component's product type, technology type, and in the setup of the component itself.

The technology type is where most settings for the Banner Launcher Service are setup and is delivered with the name “Banner Forms”. In technology types, the class that is used for launching Banner is com.sungardhe.workflow.launcher.BannerWebLauncher

Keeping Passwords in Sync between Banner Workflow and Banner

Banner Workflow requires that the same user name and password be used in both Banner Workflow and Banner. To keep the passwords in sync two methods are used, Banner External Authentication and LDAP External Synchronization.

Set Up Banner and Banner Workflow

To set up Banner with Banner Workflow, you will perform the following steps:

1. Banner Side Configuration.
   - Enable Banner Workflow in GUAINST.
   - Enable the Event Queue.
   - Set up Banner Workflow with Internet Native Banner.
   - Password Synchronization.

2. Configure technology types in Banner Workflow.


5. Import Banner Components and Examples.


**Step 1  Banner Side Configuration**

The first step in integrating Banner and Banner Workflow is typically performed by a Banner administrator.

**Enable Banner Workflow in GUAINST**

To turn on Banner Workflow integration within Banner, go to the GUAINST form and ensure that the **Workflow Enabled** check box is selected.

**Enable the Event Queue**

On the Event Queue Name Definition Form (GOREQNM), ensure that the **Active** check box is selected for all events that should start inserting records into the Event Queue table.

**Set Up Banner Workflow with Internet Native Banner**

Banner Workflow only supports integration with Internet Native Banner using servlet mode. Prior to proceeding, ensure that your Internet Native Banner installation is using servlet mode.

**Note**

All references to the FORMS_HOME directory are specific to the version the Oracle Application server being used.

1. Within your FORMS_HOME/server directory locate the formsweb.cfg file. Go to the USER PARAMETERS section and add a Runform argument:

   \[\text{wfargs=}\]

2. Modify FORMS_HOME/server/base*.htm file(s) as necessary for the browsers you use Internet Native Banner. Find the lines that match the following:

   \[
   \text{<PARAM NAME="serverArgs" VALUE="module=\%form\%"} \\
   \text{and} \\
   \text{serverArgs="module=\%form\%"}\]

3. Update the lines to match the following:

   \[
   \text{<PARAM NAME="serverArgs" VALUE="module=\%form\% %wfargs\%"} \\
   \text{and} \\
   \text{serverArgs="module=\%form\% %wfargs\%"}\]
With this modification, the generated html form that holds the JInitator applet code will contain dynamic workflow arguments when being initiated by Banner Workflow.

**Note**

Place `%wfargs%` at the very end of your "serverArgs" parameter.

---

**Password Synchronization**

Banner Workflow requires that the same user name and password be used in both Banner Workflow and Banner. To keep the passwords in sync, there are two methods to use.

**Banner External Authentication Plug-in**

Banner Workflow can validate its user name and password against a Banner database for authentication. Essentially there will be no syncing of passwords since the password being used in Banner Workflow is the actual Banner password.

To setup external authentication to a Banner database, make the following changes in the `WORKFLOW_HOME/config/configuration.xml` file:

1. Find the `ExternalAuthenticator` section of the configuration.xml file.

2. Modify the section to look like what's listed below.

   ```xml
   <Authentication mode="WorkflowExternal">
   <WorkflowExternal>
   <ExternalAuthenticator>
   <ClassName>com.sungardhe.workflow.security.BannerAuthenticator</ClassName>
   <Properties/>
   </ExternalAuthenticator>
   </WorkflowExternal>
   </Authentication>
   ```

3. After saving the changes, go to the `WORKFLOW_HOME` directory and run `bin/wftool uploadconfig` to commit the configuration changes to the database.

4. Restart the OC4J instance tied to the `WORKFLOW_HOME`.

In changing the external authentication, the ability to change a password in Banner Workflow is now disabled. If a user would like to change their password they'll have to go into Banner or have a DBA alter the password.

**Note**

When using the Banner Authentication, Banner Workflow performs username/password authentication by attempting to open a jdbc connection to the Banner database with the username/password. For more information, see “Authentication” on page 143.
LDAP External Authentication Plug-in

For those who have Luminis installed and would like to have Luminis manage user identities, Banner Workflow provides a plug-in to authenticate against the Luminis LDAP server. See “Configuring Banner Workflow for Luminis SSO” on page 77 for more information.

Step 2 Configure Technology Types in Banner Workflow

To ensure that Banner Workflow can launch Banner, you must configure the Banner Forms technology type. To do this, you need to log in to Banner Workflow as an administrative user and modify the Banner Forms technology type.

To modify the Banner Forms technology type:

1. Select Workflow System Administration from Administration.
2. Select Technology Types from the Workflow System Administration page.
3. Select Banner Forms.

Set the Launch Service

The default value for the Web Launch Service Name for Banner is `com.sungardhe.workflow.launcher.BannerWebLauncher`.

Set Launch Parameters

The following steps have to be applied. From the Banner Forms technology type page select the launch parameters.

Set up the following values:

- `banner_host_string` - the host string used to log onto Banner. Typically this is the same value as the Oracle SID for the particular database.
- `banner_config` - the name of the config to use for the Internet Native Banner configuration. The config for an Internet Native Banner environment is defined in the formsweb.cfg file.
- `banner_inb` - the URL used to start Internet Native Banner. The value is the fully qualified URL with either HTTP or HTTPS. For example:
  - If the Internet Native Banner server was running on machine `banner.sungardhe.com` and its servlet was located at `/servlet/f90servlet` then the value would be:
  
  `http://banner.sungardhe.com/servlet/f90servlet`
• *banner_timeout* - the length of time in seconds that the Banner Workflow integration service will allow a single sign on to occur from Banner Workflow to Banner. In most situations the default of 30 seconds should be sufficient.

The *banner_username* and *banner_password* will typically be unchanged and should be *WFSSO* and *u_pick_it* respectively. As discussed in the technical overview, the WFSSO user account is to be created solely to provide single sign on ability. The account is not considered secure and should not have any more grants than what is specified in the WFSSO create user script.

### Step 3 Setup Event Processing in Banner Workflow

An Event Provider is needed to retrieve event information from each event source, for example, Banner.

To simplify event dispatching configuration, the Configuration Wizard provides the ability to copy already set up Banner configuration settings into the event dispatcher setting. To run the wizard:

1. Go to your WORKFLOW_HOME directory and run `configwizard`.
2. Select **Banner Event Dispatcher**.
3. Select Y to copy the JDBC connection information from the J2EE data source that was set up previously.
4. Select the user name and password to be used to connect to Banner event tables. This value is typically *WFEVENT*.
5. Once back to the main window ensure that your changes are saved by selecting **S**.

After running the wizard you should upload the configuration to the workflow configuration tables. To upload this file, run `bin/wftool uploadconfig` from your WORKFLOW_HOME directory.

Information on the settings for the event provider in the configuration.xml file can be found in the table below:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the event provider, e.g. <em>Banner Workflow Event Provider</em>. Used in the log files.</td>
</tr>
<tr>
<td>classname</td>
<td>Java class for the Banner event provider: com.sungardhe.workflow.engine.externalevent.provider.BannerTableExternalEventProvider</td>
</tr>
</tbody>
</table>

January 2009
### Property Description

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enabled</td>
<td>Turns the event dispatcher on or off without removing it from configuration.xml. If this value is changed, just as the case with other changes to configuration.xml, the file must be uploaded into the Banner Workflow system and then both the OC4J instance(s) and engine(s) need to be restarted.</td>
</tr>
<tr>
<td>pollingInterval</td>
<td>Amount of time the event provider delays between polls of the event table, in seconds. The default time is 10 seconds.</td>
</tr>
<tr>
<td>jdbcDriver</td>
<td>Class used for JDBC connections to the database: The default value is <code>oracle.jdbc.driver.OracleDriver</code> and should not be changed unless instructed by SunGard Higher Education.</td>
</tr>
<tr>
<td>connectionURL</td>
<td>JDBC connection URL. For example: <code>jdbc:oracle:thin:@sunhost1:1521:B60</code></td>
</tr>
<tr>
<td>user</td>
<td>User ID used to connect to the database. The user should have read and write access to the Banner event tables.</td>
</tr>
<tr>
<td>password</td>
<td>Password used to connect to the database.</td>
</tr>
<tr>
<td>maxResultsPerPoll</td>
<td>Maximum number of event records to retrieve during each poll. The default value is 50, so on every poll the Banner Event Provider will retrieve up to 50 events.</td>
</tr>
<tr>
<td>targetName</td>
<td>Name of Event Target used by the Banner Workflow Event Provider. This value in most cases is set to the default value of <code>WORKFLOW</code>.</td>
</tr>
<tr>
<td>Source ID</td>
<td>Specifies the source of the External Events handled by this provider. All External Events posted to workflow are required to have a source ID that uniquely identifies the external system generating the events. See “External Events” on page 139 on source IDs and event IDs.</td>
</tr>
</tbody>
</table>

By default, the Banner Event Provider will poll at least every 10 seconds and retrieve at most 50 events at a time, totaling a maximum of 300 events a minute. If more than 300 events a minute are firing, the “maxResultsPerPoll” element can be increased to handle more events at a single time with the understanding that overall system performance may be impacted.
Step 4  Setup Automated SQL Activities in Banner Workflow

Most workflows will require the ability to automatically retrieve data from Banner and bring those values into Workflow Context. To do so, a Workflow Data Source must be set up in the configuration.xml file.

To simplify the data source configuration, the Configuration Wizard provides the ability to copy down already setup Banner configuration settings into the data source settings. To run the wizard:

1. Go to your WORKFLOW_HOME directory and run bin/configwizard.

2. Select Banner Automated Activity Data Source.

3. Select Y to copy the JDBC connection information from the J2EE data source that was set up previously.

4. Select the user name and password to be used to query Banner tables and call Banner stored procedures from automated activities.

   This value is typically WFBANNER or WFAUTO.

**Note**

You may need to add additional grants to the account used here depending on your usage.

5. Once back to the main window ensure that your changes are saved by selecting S.

Below is an example of the BannerDatabase in configuration.xml.

```xml
<DataSources>
 <DataSource name="BannerDatabase">
  <Url>jdbc:oracle:thin:@banner_server_name:1521:banner_sid</Url>
  <Username>wfauto</Username>
  <Password>u_pick_it</Password>
 </DataSource>
</DataSources>
```

After saving changes to the configuration.xml file, go to WORKFLOW_HOME/bin and run wftool uploadconfig to upload the newly modified configuration.xml up to the Banner Workflow application running on the application server. After uploading the file, stop and then start the OC4J instance on the Oracle Application Server that pertains to this installation of Banner Workflow.
Step 5  Import Banner Components and Examples

Banner Workflow provides seed data to be imported into an environment that is integrated with Banner. The seed data will provide Business Components that represent Banner forms as well as examples that use these components and demonstrate using Banner Workflow with Banner.

The import files are located in the support jar available off the ActionLine or off of the support directory on the Banner Workflow CD.

Please see “Import and Export Tools” on page 88 for more information.

For Banner 7.x the files are located in support/SCTBanner/7.0 and should be imported in the order listed below. With Banner 7 there are three import files:

1.  Components_1.xml - Provides all of the categories and components needed to integrate Banner Workflow with a Banner 7 environment.

2.  WorkflowExamples_2.xml - Provides the examples that are designed for Banner 7.

3.  MifTypes.xml - Contains technology types for creating MIF aware SQL Stored Procedures/Queries. This xml file Should only be loaded if you are integrating with a Banner instance that is using the Multi Institution Functionality.

Step 6  Extract Banner User Data and Import Into Banner Workflow Database

You can optionally extract user information from the Banner database into a text file and import the text file into the Banner Workflow database. If you bypass this step, you must add each user to the Banner Workflow database manually.

User Information That Is Extracted

This step extracts the following user information into a text file:

<table>
<thead>
<tr>
<th>Information</th>
<th>Source of information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oracle username</td>
<td>GOBEACC table</td>
</tr>
<tr>
<td>(in lower case)</td>
<td></td>
</tr>
<tr>
<td>First and Last names</td>
<td>SPRIDEN table</td>
</tr>
<tr>
<td>Password</td>
<td>Manual entry when the extract script is run</td>
</tr>
<tr>
<td>Enabled status</td>
<td>Automatically set to true</td>
</tr>
</tbody>
</table>
### Extract and then Import User Information

1. Ensure that the SPRIDEN table contains information for all users. As a minimum, the first or last name is required.

2. Use the Enterprise Access Control Form (GOAEACC) to associate each Oracle username with a Banner user ID.

3. Make sure that a record exists on the Crosswalk Validation Form (GTVSDAX) with the following information:

<table>
<thead>
<tr>
<th>Internal Code</th>
<th>Source of information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Internal Group</td>
<td>WORKFLOW</td>
</tr>
<tr>
<td>External Code</td>
<td>EMAILTYPE</td>
</tr>
<tr>
<td>Desc</td>
<td>Email Code</td>
</tr>
</tbody>
</table>

   This record identifies the type of email addresses you want to extract.

4. Make sure that each user ID in the GOREMAL table has an email address with the email type code to be extracted.

5. Use SQL*Plus to run the user extraction script WORKFLOW_HOME\examples\banner\XMLCreation\wfaruser.sql. When you are prompted for a default password, enter a password without quotation marks. This password will be assigned to all users.

6. When the script is finished, review the WF_Users.xml file for errors.

   If necessary, correct any errors and run the extract process again.

When the text file is error-free, import the text file into the Banner Workflow database. For more information, see “import” on page 158.
Step 7  System Verification

At this point all modifications needed to integrate Banner Workflow with Banner should be completed.

To assist in determining if an installation is valid, a simple workflow was created for testing purposes. The system verification workflow definition is a small workflow designed to test various integration points in the Banner Workflow system. It assumes that your have already loaded the base data into the system, have setup all of your data sources and the email server, and are using the standard names for the Technology Types, Product Types, and DataSources used to integrate with Banner. If this is not the case, you will need to edit the components and the workflow accordingly.

Note
You must edit the Banner Forms technology type to correspond with your current Banner installation. For more information please see “Configure Technology Types in Banner Workflow” on page 53.

You must import the verification bootstrap in order to get access to this special workflow. Following the import instructions execute the following command:

```bash
WORKFLOW_HOME/bin/import wfroot <password> WORKFLOW_HOME/bootstraps/SystemVerification.xml
```

To run the workflow, use the modeler to load and verify the workflow definition, then move it to Test mode and start it. After you have successfully completed the workflow, you can leave it in your system or remove it. If you choose to remove it, simply delete it from the modeler, remove the two components in the system verification category, then remove the system verification category itself.

Analyze Banner Components

System administrators can use Analyze Banner Components to assist in the process of migrating and updating workflows that use components associated with an older Banner release to a new release. This tool compares the current system's Banner components with those packaged in a bootstrap. It reports which Banner components have parameters that have changed and the workflows where the affected components are used.

Typically the Analyze Banner Components tool is used to compare the impact of a major release of Banner to the already established workflows in a Banner Workflow system. For example, if you have 2 workflows in your system that use 10 Banner forms, you would want to check to see if a new release of Banner would impact your workflows. You may check to see what impact a new release of Banner will have on your workflows by supplying the Banner Workflow environment with the Components_1.xml file from the support jar or directory. The Components_1.xml file contains all of the components that relate to a release of Banner.
To analyze your Banner components:

1. Select **Workflow System Administration** from Administration.

2. Select **Analyze Banner Components** from the Workflow System Administration page.

3. Click **Browse** and locate the bootstrap that contains the updated Banner components. The bootstrap file can have an XML or ZIP suffix. Typically this file is the `Components_1.xml` file from the support jar or directory.

4. Select your analysis options. By default, the analysis will be only run against Banner components that are used in an existing workflow definition. The report will list, by workflow, the changed components. Depending on the number of components, the component analysis may take several minutes to run.

5. Click **Analyze** to begin the process.

To compare Banner components, the component analyzer performs two-levels of comparisons. First, the analyzer tries to find the newer version of a component so that it can be compared against the current system's component. When comparing a component, if the analyzer is unable to find a component in the bootstrap with the same exact name, it will try to find it by either truncating `Form` from the previous component's name or finding components that share the same Banner form name (same `formName` parameter value). Once the newer version of the component is found in the bootstrap, it will be compared with the current system's component to determine which parameters have been modified, added, or removed.

After the analysis completes, depending on the options selected, a framed HTML report is displayed. In the left-hand column, there is a navigation and a summary area. The summary provides a final count for the number of affected workflow definitions or components. The navigation provides links to jump directly to a workflow definition or Banner component. In the report, under each component, there is a parameter listing.

<table>
<thead>
<tr>
<th>Difference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Added</td>
<td>The current system's component did not have this parameter.</td>
</tr>
<tr>
<td>Removed</td>
<td>The parameter has been removed from the newer component.</td>
</tr>
<tr>
<td>Modified</td>
<td>The parameter's type, required, or guaranteed status has been modified.</td>
</tr>
</tbody>
</table>

To print the report for offline viewing and analysis, click Print.
**Update Workflows for a New Banner Release**

The following are suggested steps for using Analyze Banner Components to update workflows for a new Banner release:

1. Run *Analyze Banner Component* with the default options.

2. Print the report.

3. After considering the impact of moving to a new release of Banner based off of the report, import the Banner support files as listed in “Import Banner Components and Examples” on page 57.

4. Use the report to open up each affected workflow. You should create a new version of the affected workflow and then adjust the affected activities to address the changes. Ensure that the components being used are the newly imported components from step 3. After making changes, use the validation feature from within the Workflow Modeler to ensure that your changes are valid.

**Handling Custom Forms**

Before using custom forms in your workflows, you must make sure your custom forms are Workflow-enabled so they will behave as expected when invoked as workflow tasks and activities. You should Workflow-enable the forms that you create to meet your specific needs, as well as Banner baseline forms that you modify.

These are the basic steps involved in Workflow-enabling custom forms. Some of these steps do not require any action, but are simply listed as possible issues that you may solve programmatically before using that form in a workflow.

1. Verify that the form uses Banner Release 7.0 standards.

2. Check for a *WHEN-TIMER-EXPIRED* trigger.

3. Define the form to *Banner Workflow*.

4. Check navigation of key block items.

5. Check visual attributes of key block items.

6. Check triggers of key block items.

7. Check the *PRE_FORM_TRG*.

8. Check the startup logic.
9. Add logic if the form is a pass-through form.

10. Check for automatic rollbacks.

11. Prevent automatic commits in *Workflow Submit*.

12. Make changes for additional *Workflow Submit/Release* logic.

The rest of this chapter describes each step in detail.

**Step 1  Verify That the Form Uses Banner Release 7.0 Standards**

To communicate properly with Banner Workflow, a custom form must be converted to Banner Release 7.0 standards. See the Banner 7.0 UI Methodology Handbook ([meth70000hb.pdf](https://udc-support-center.com)) on the UDC Support Center for further instructions.

When you convert a custom form to 7.0 standards, ensure the GOQWFLW library is attached. This library contains all of the required functions for Banner Workflow enabling Banner. For more information on the GOQWFLW library and its interaction with each form, refer to “[Banner Workflow-Awareness Library (GOQWFLW)](https://udc-support-center.com)” on page 69.

**Step 2  Check for a WHEN-TIMER-EXPIRED Trigger**

Check to see whether the form has a WHEN-TIMER-EXPIRED trigger. If it does, add the following logic to it:

```plaintext
IF GET_APPLICATION_PROPERTY(TIMER_NAME) = G$_WF_DRIVER.WF_TIMER_NAME THEN G$_WF_DRIVER.WF_EXECUTE('WF_CHECK_MSG'); END IF;
```

This logic makes it possible, in a future release, for every Banner form to listen for and respond to requests to perform workflow tasks and activities. Specifically, this logic enables any form, upon the expiration of a timer created for Banner Workflow, to launch the appropriate Banner object that is associated with a requested workflow task or activity.

**Step 3  Define the Form to Banner Workflow**

For Banner Workflow to launch a custom form as an activity, the form must be defined as a valid business component to Banner Workflow. The component is defined in the Business Component Catalog (BCC).
If your form is a modified version of an existing baseline Banner form with the same name, you may be able to use the existing baseline component definitions under the following conditions:

- If you do not need to define additional parameters or change existing ones (defined in block.item format) for the form, you can use the existing baseline component.
- If you need to define additional parameters or change existing ones so that the appropriate workflow context is passed into and out of your form in any workflow you create, you can copy the baseline definition for the form in the BCC and modify the copy as appropriate.

If your form was created locally and does not have the same name as any existing baseline Banner form, you can copy and rename as appropriate the existing component definition for any baseline Banner form within the BCC and modify the definition as follows:

- Use an executable ID of the name of your local form as defined within the General Object Base Table (GUBOBS) for that form.
- Reflect the parameters (in block.item format) that must be passed in and out of the appropriate workflow context.

For more information on using the Business Component Catalog, refer to the Banner Workflow Analyst/Administrator Handbook.

**Step 4 Check Navigation of Key Block Items**

If the form contains any key block items to which a user (or the form) can navigate without performing a rollback, move those items to another block (for example, FORM_HEADER). Update the logic of the form as appropriate.

If this change is not made and a user navigates from a data block to the key block without performing a rollback (for example, moves to a pop-up window that displays navigable key block items), GOQWFLW still assumes that a rollback was performed and that the items are enabled and runtime errors will result.

**Step 5 Check Visual Attributes of Key Block Items**

If the form contains any key block items with custom visual attributes:

- Update the items to use an existing named attribute.
  
  or

- Define new named visual attributes in the form (or in a referenced form) for those custom attributes. Update the items to use the new named attributes.
Oracle Forms does not allow you to reapply a custom visual attribute for an item that was established at design time. You can only reapply a named visual attribute (for example, 'G$_NVA_TEXT_ITEM') or NULL (which restores the default). Therefore, a key block item that was populated with a parameter value, protected, and highlighted, loses its custom visual attributes when the highlight is removed from the item if the user performs a Workflow Release.

**Step 6  Check Triggers of Key Block Items**

If the form contains any *WHEN-<ITEM_TYPE>-CHECKED/CHANGED* triggers for a check box, radio group, or list item in the key block, review the following information to determine what, if any, changes should be applied.

Item-type specific *WHEN-<ITEM_TYPE>-CHECKED/CHANGED* triggers only fire when the user selects a value from a pull-down list item, selects a radio button, or selects/deselects a check box. These triggers do not fire when an item of that type is programmatically updated. For example, these triggers do not fire if the values of their associated items are updated by a global copy upon entering a form, the return of a value from another form via the Select function, the entry of an item with an input parameter value through the GOQWFLW library, or any other programmatic way.

Use these steps to make the necessary changes:

1. Open the form that contains the check box, radio group, or list item in Forms Designer.

2. Determine whether a form-level, block-level, or item-level *WHEN-<ITEM_TYPE>-CHECKED/CHANGED* trigger would fire when the item value is changed.

   If there is no such trigger, stop here.
   
   or
   
   If there is a such a trigger, proceed to step 3.

3. Examine the logic of the *WHEN-<ITEM_TYPE>-CHECKED/CHANGED* trigger.

   If the logic will fire only when a user physically changes the value of the item (versus a change made by the application program), stop here.
   
   or
   
   If the logic will also fire when the application program changes the value of the item, proceed to step 4.
4. Determine whether there is a form-level, block-level, or item-level WHEN-VALIDATE-ITEM or POST-CHANGE trigger that would ever fire for the item.  

If there is no such trigger, skip to step 6.  

or  

If there is such a trigger, proceed to step 5.  

5. Examine the logic of the trigger:  

   • If the logic clones the WHEN-<ITEM_TYPE>-CHANGED/CHECKED trigger, and you are comfortable with the same logic potentially executing twice, stop here.  

   • If the logic clones the WHEN-<ITEM_TYPE>-CHANGED/CHECKED trigger, and you are not comfortable with the same logic potentially executing twice, remove all logic from the WHEN-<ITEM_TYPE>-CHANGED/CHECKED trigger and skip to step 7.  

   • If the logic does not clone the WHEN-<ITEM_TYPE>-CHANGED/CHECKED trigger, and you are comfortable with WHEN-<ITEM_TYPE>-CHANGED/CHECKED trigger logic always executing when WHEN-VALIDATE-ITEM or POST-CHANGE fires, then copy the logic of the WHEN-<ITEM_TYPE>-CHANGED/CHECKED trigger to the WHEN-VALIDATE-ITEM or POST-CHANGE trigger. Then do one of the following:  

     If you are comfortable with the same logic potentially executing twice, stop here.  

     or  

     If you are not comfortable with the same logic potentially executing twice, delete the logic from WHEN-<ITEM_TYPE>-CHANGED/CHECKED that you just moved to the WHEN-VALIDATE-ITEM or POST-CHANGE trigger, and skip to step 7.  

   • If the logic does not clone the WHEN-<ITEM_TYPE>-CHANGED/CHECKED trigger, and you are not comfortable with WHEN-<ITEM_TYPE>-CHANGED/CHECKED trigger logic always executing when WHEN-VALIDATE-ITEM or POST-CHANGE fires, stop here.  

6. Create a new WHEN-VALIDATE-ITEM trigger at the scope where the WHEN-<ITEM_TYPE>-CHANGED/CHECKED trigger that you located in step 2 resides. Move the logic contained within the WHEN-<ITEM_TYPE>-CHANGED/CHECKED trigger to this new trigger.
7. Update the WHEN-\(<ITEM\_TYPE>\)-CHANGED/CHECKED trigger to execute only the following statement:

\[
\text{VALIDATE(ITEM\_SCOPE);}
\]

These changes force the execution of any POST-CHANGE or WHEN-VALIDATE-ITEM triggers for that item, along with any default Oracle Forms validation processing whenever a user manually changes the value of a check box, radio group, or list item as long as the Mouse Navigate property for that item is set to True.

If the Mouse Navigate property of the item cannot be set to True for functional or technical reasons, then consider saving off the name of the \textit{SYSTEM\_CURSOR\_ITEM}, performing a \textit{GO\_ITEM} to the check box, radio group, or list item, executing \textit{VALIDATE(ITEM\_SCOPE)}, and then returning the user to the saved off value of \textit{SYSTEM\_CURSOR\_ITEM}. This is what the GOQWFLW library does when it populates key block items.

**Step 7  Check the PRE\_FORM\_TRG**

Check the forms's \textit{PRE\_FORM\_TRG} trigger, or any triggers that are fired by it, for logic that displays alerts. Move any such logic to a WHEN-\textit{NEW-FORM-INSTANCE} trigger or a similar trigger that fires after \textit{PRE\_FORM\_TRG}. By making this change, the Banner form will come into focus as appropriate when it is invoked as a workflow task or activity.

If this logic is not moved within the form, the logic executed by GOQWFLW when a form's \textit{PRE-FORM} trigger fires will fail when it tries to bring the MDI application window of Banner into focus. This results in the need to toggle between Banner and Banner Workflow each time you want to perform the workflow task or activity related to the form.

**Step 8  Check the Startup Logic**

Check to see whether you need to perform some logic at the startup of the form if it is currently considered a workflow task or activity. If so, add the startup logic to the WHEN-\textit{NEW-FORM-INSTANCE} trigger or some other trigger that fires after \textit{PRE\_FORM\_TRG} within the form.

\[\text{Note}\]

Do not add startup logic to \textit{PRE\_FORM\_TRG}. GOQWFLW logic does not determine whether a form is a workflow task or activity until after \textit{PRE\_FORM\_TRG} is executed. Therefore, the part of the startup logic that evaluates whether the current form is a workflow task or activity would be inaccurate.
Step 9  Add Logic if the Form Is a Pass-Through Form

Determine whether either of the following conditions ever exists for the form:

- The user must explicitly exit the form before proceeding to the form that the user actually chose.
- The user must explicitly acknowledge an alert or message before proceeding to the form that the user actually chose.

If either of these conditions exists, add the following lines of code to the form's logic where the form comes into focus when the user is stopped before navigating to the form that is the actual workflow task or activity to be performed:

```java
IF G$_WF_CONDITIONS.IS_WF_PASSTHROUGH_FORM THEN
  G$_WF_SET_FOCUS.SET_FOCUS;
END IF;
```

Without this logic, the MDI application window of Banner will not automatically come into focus. You would have to manually toggle from Banner Workflow to Banner to see that your activity did not really fail to launch, but that you were temporarily taken to another form to perform an activity.

Note
This step only applies to Banner 6.x sites.

Step 10  Check for Automatic Rollbacks

Check to see whether any functions (for example, Save) cause the form to perform an automatic rollback or clear values that are needed by Banner Workflow before the user is able to perform a Workflow Submit. These types of automatic rollbacks will cause problems. When the user selects Workflow Submit to communicate back to Banner Workflow, the rollback has already cleared the data. Consequently, null context parameter values will be passed to Banner Workflow.

To correct this problem, you can either change the logic to no longer clear the values if this is functionally acceptable, or you can use the following steps to change the form so it performs an implicit Workflow Submit when the particular function is invoked:

1. Locate the trigger that fires the rollback (clear form). Usually this can be done by finding the function that performs the action and tracing the triggers that it executes.

2. Locate the rollback action. This is usually at the end of the logic after all of the edits have fired to check whether the action can be performed. In most cases, it is a key-clrfrm, not a rollback.

In the following example, the rollback action is a call to a user-defined trigger called `clr_form`:

```java
EX-<< PASS_COMMIT >>
EXECUTE_TRIGGER( 'clr_form' ) ;
```
3. Modify the rollback action appropriately.

In the following example, if the form is called as a workflow activity, an implicit Workflow Submit is performed instead of the rollback logic.

```sql
Ex-<< PASS_COMMIT >>
IF G$_WF_CONDITIONS.IS_WF_ACTIVITY
AND NOT G$_WF_CONDITIONS.IS_WF_SUBMITTED THEN
G$_WF_CONTROL_FORM.WF_SET_COMMIT_OVERRIDE;
G$_WF_CONTROL_FORM.WF_SUBMIT:
else
EXECUTE_TRIGGER( 'clr_form' ) :
end if;
```

4. Change other informational messages and autohints as necessary to communicate the behavioral change of the form when it is a workflow activity.

**Step 11 Prevent Automatic Commits in Workflow Submit**

Determine whether there is any scope or condition, other than the pre-existing conditions listed below, when the form should not automatically perform a commit when a user selects the Workflow Submit function. If there is any such scope or condition, you can use the following procedures inside a form to override commits:

- When you want a commit to be skipped, add a call to the procedure `G$_WF_CONTROL_FORM.WF_SET_COMMIT_OVERRIDE`.
- When you no longer want a commit to be skipped, add a call to the procedure `G$_WF_CONTROL_FORM.WF_RESET_COMMIT_OVERRIDE`.

Logic is already in place to prevent a commit from being performed for the following pre-existing conditions:

- The form name indicates that it is an inquiry-only form. Specifically, if the third letter of the form name is C or I, or if the first three letters are FTV, the form is considered an inquiry form. This condition excludes query (Q) forms, because they cannot be launched directly from the main menu and therefore cannot be launched as workflow activities.
- The role of the current user, granted either directly or via a class, only allows the user to query the form.
- The form contains no new or changed records.
Step 12  Make Changes for Additional Workflow Submit/Release Logic

If the form has a scope or condition that requires a Workflow Submit or a Workflow Release to perform additional logic, make the following changes:

- Add a local `WF_SUBMIT_TRG` or `WF_RELEASE_TRG` to the appropriate scope.
- Add a call to the procedure `G$ WF_CONTROL_FORM.WF_SET_LOCAL_TRG_EXISTS('WF_SUBMIT')` or `('WF_RELEASE')` to notify Banner to execute the trigger when a Workflow Submit or a Workflow Release is performed.
- Add a call to `G$ WF_CONTROL_FORM.WF_RESET_LOCAL_TRG_EXISTS ('WF_SUBMIT')` or `('WF_RELEASE')` to notify Banner to not execute the trigger when a Workflow Submit or a Workflow Release is performed.

For an example of changes for a Workflow Submit, see these GLRSLCT form-level triggers:

```
WF_SUBMIT_TRG
WHEN-NEW-FORM-INSTANCE
```

For an example of changes for a Workflow Release, see these GJAPCTL triggers:

```
WF_RELEASE_TRG (form level)
WF_RELEASE_TRY (KEY_BLOCK)
PRE-BLOCK (KEY_BLOCK)
```

Technical Information

This section provides technical information on the Banner Workflow-Awareness Library (GOQWFLW).

Banner Workflow-Awareness Library (GOQWFLW)

This library is the single repository for all baseline cross-product Banner Workflow functionality available within Banner when it is invoked from Banner Workflow. To guarantee that the required Banner Workflow functionality can be accessed within each form, this library is attached to every baseline Banner form that is defined as a component to Banner Workflow.

**Note**

It is highly recommended that you attach the GOQWFLW library to all forms, even if you are not using Banner Workflow.
**What Is the Relationship Between GOQWFLW and Banner Workflow?**

The GOQWFLW library provides a PL/SQL interface between Banner and Banner Workflow.

For example, through the GOQWFLW interface to Banner Workflow, a Banner session can:

- Determine whether a user has requested to perform a workflow activity.
- Determine the corresponding form or process to open to perform the activity.
- Retrieve and return the parameters (if any) that were defined for that activity.
- Notify Banner Workflow when the user has completed the activity.

**How Does a Form Invoke the Functionality of GOQWFLW?**

For a Banner form to access the functionality provided by GOQWFLW, it is not enough to attach this library to the form. Instead, GOQWFLW must control the behavior of the form, which has been invoked as an activity, when certain events occur within the form.

Forms that comply with Banner Release 7.0 standards have the appropriate triggers, within the GS_FORM_CLASS of GOQOLIB, that call the required functions and procedures of GOQWFLW as they should. Forms that do not comply with Release 7.0 standards most likely reference their own local copies of the triggers and don't execute the necessary GOQWFLW logic. These forms do not behave as they should when invoked as activities.

When a user first enters a form, compliance with Release 7.0 standards guarantees that the PRE-FORM trigger of the GS_FORM_CLASS of GOQOLIB fires. This in turn eventually executes the GS_WF_CONTROL_FORM.CONTROL_ENTRY procedure within GOQWFLW. This procedure registers the current form as an activity and brings the form into focus if the user has in fact requested the activity.

A form can be individually modified to take advantage of other functions and procedures provided by GOQWFLW as necessary. (See “Handling Custom Forms” on page 61 for more information.) The majority of forms, however, should not require modifications if they comply with the Banner Release 7.0 standards. These standards ensure that GOQWFLW can exhibit control in a form when needed.
**How Is GOQWFLW Organized?**

With the exception of the procedure `AUDIT_TRAIL_4_0`, that contains the audit trail for GOQWFLW, the Banner Workflow-Awareness Library is organized into a group of packages with distinct purposes. Each package, and each procedure and function in that package, is documented to some extent within the library itself.

The following table describes the function of each package in GOQWFLW:

<table>
<thead>
<tr>
<th>Package</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>G$_WF_BA</td>
<td>Provides a PL/SQL interface to communicate with Banner Workflow.</td>
</tr>
<tr>
<td>G$_WF_CONDITIONS</td>
<td>Contains functions that evaluate conditions and return Boolean values to control the behavior of Banner as it polls for workflow tasks and activities to perform, and as it executes those tasks and activities when they are launched from Banner Workflow.</td>
</tr>
<tr>
<td>G$_WF_CONTEXT_PROCESS_IN</td>
<td>Provides the interfaces needed to:</td>
</tr>
<tr>
<td></td>
<td>• Obtain parameter names and values from Banner Workflow.</td>
</tr>
<tr>
<td></td>
<td>• Populate and repopulate the items of a form with parameter values as appropriate.</td>
</tr>
<tr>
<td></td>
<td>• Delete the parameter names and values when they are no longer needed.</td>
</tr>
<tr>
<td>G$_WF_CONTEXT_PROCESS_OUT</td>
<td>Provides the interfaces needed to:</td>
</tr>
<tr>
<td></td>
<td>• Extract parameter names and (potentially updated) values from Banner.</td>
</tr>
<tr>
<td></td>
<td>• Communicate any updated parameter values to Banner Workflow.</td>
</tr>
<tr>
<td>G$_WF_CONTEXT_VALIDATE</td>
<td>Contains functions that return Boolean values to indicate the status of the:</td>
</tr>
<tr>
<td></td>
<td>• Names or values of the parameters for a workflow activity.</td>
</tr>
<tr>
<td></td>
<td>• Key block items of a form that have been or will be populated with input parameter values.</td>
</tr>
<tr>
<td>Package</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>G$<em>WF_CONTROL</em></td>
<td>Provides the interfaces needed to control the behavior of a form that has been launched as a workflow task or activity. Specifically, this package controls the behavior of a form when:</td>
</tr>
<tr>
<td>FORM</td>
<td>• A form is entered <em>(PRE-FORM trigger has fired)</em>.</td>
</tr>
<tr>
<td></td>
<td>• A block is entered <em>(WHEN-NEW-BLOCK-INSTANCE trigger has fired)</em>.</td>
</tr>
<tr>
<td></td>
<td>• The Workflow Release function is executed <em>(G$_WF_BUTTON_PRESSED_TRG trigger has fired)</em>.</td>
</tr>
<tr>
<td></td>
<td>• The Workflow Submit function is executed <em>(G$_WF_BUTTON_PRESSED_TRG trigger has fired)</em>.</td>
</tr>
<tr>
<td></td>
<td>• A form is exited <em>(POST-FORM trigger has fired)</em>.</td>
</tr>
<tr>
<td>G$<em>WF_CONTROL</em></td>
<td>Provides the interfaces needed to control the behavior and appearance of a form's key block items that are populated with input parameter values from Banner Workflow.</td>
</tr>
<tr>
<td>KEY_ITMS</td>
<td></td>
</tr>
<tr>
<td>G$<em>WF_DELIMITED</em></td>
<td>Contains functions and procedures that parse and handle delimited strings. These strings have values that are separated in the following type of format: G$_WF_DRIVER</td>
</tr>
<tr>
<td>STRING</td>
<td>&lt;field_1&gt;&lt;delimiter&gt;&lt;field_2&gt;&lt;delimiter&gt;...</td>
</tr>
<tr>
<td>G$_WF_DRIVER</td>
<td>Provides the interface for implementing all Workflow-specific procedures and functions that need to be consistently called outside the GOQWFLW library.</td>
</tr>
<tr>
<td></td>
<td>This implementation reduces the need to regenerate all forms when the full Banner Workflow release of Banner is delivered.</td>
</tr>
<tr>
<td>Package</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>G$_WF_ERROR</td>
<td>Provides the interfaces needed to:</td>
</tr>
<tr>
<td></td>
<td>• Present the user with consistently formatted error messages when an error occurs in a workflow task or activity.</td>
</tr>
<tr>
<td></td>
<td>• Help the user debug a task or activity if he or she needs to know the parameters under which a task or activity is operating.</td>
</tr>
<tr>
<td>G$_WF_HEADER</td>
<td>Declares the values of all public constants that may be used by other GOQWFLW packages and by other Banner Oracle forms modules as appropriate.</td>
</tr>
<tr>
<td>G$_WF_ICONS</td>
<td>Provides the interfaces necessary to control the behavior and display attributes of the Banner Workflow specific iconic buttons and the Banner Workflow specific form pulldown menu items.</td>
</tr>
<tr>
<td>G$_WF_RECGRP</td>
<td>Provides the interfaces necessary to readily retrieve information from record groups.</td>
</tr>
<tr>
<td>G$_WF_SET_FOCUS</td>
<td>Provides the interfaces necessary to control the behavior and appearance of the MDI (Multiple Document Interface) application window of Banner.</td>
</tr>
<tr>
<td>G$_WF_WAIT_FOR_WORK</td>
<td>Provides the interfaces necessary to enable a Banner session to:</td>
</tr>
<tr>
<td></td>
<td>• Launch the appropriate Banner object when a workflow task or activity is started.</td>
</tr>
<tr>
<td></td>
<td>• Create, initialize, interrogate, reset, and destroy as appropriate the constructs a Banner session uses to perform this function.</td>
</tr>
</tbody>
</table>
Integrating Banner Workflow with Luminis

Banner Workflow provides a channel interface to be rendered in the Luminis portal. With Banner Workflow 8.0, Worklist, Alert, and My Processes channels are available. For example:

To integrate Banner Workflow with Luminis the following software is required:

- Luminis Platform 4.1.1 with the latest hot fix available from Luminis Support. The most recent hot fix will ensure the CPIP connection between Banner Workflow and Luminis can be established and that when CAS or IDM is enabled the UDC identifier can be passed from Luminis to Banner Workflow.

- Banner Workflow 8.0 setup in one of the following two modes:
  - Banner Workflow 8.0 in Workflow External authentication mode, pointing to the LDAP server that is used by Luminis
  - Banner Workflow 8.0 in CAS or IDM Gateway mode
Configuring Luminis for Workflow External Authentication

This section contains the steps needed to integrate Banner Workflow with Luminis. There are two main touch points in the configuration between the two products:

- Banner Workflow in Workflow External authentication mode uses the CPIP protocol to establish single sign on from Luminis to Banner Workflow.
- Banner Workflow channels use HTTP/HTTPS to make connections from a channel in Luminis to the Banner Workflow server.

To establish single sign on from Luminis to Banner Workflow, the Banner Workflow server must be configured to use the LDAP server that Luminis is using and must be able to map the accounts in Banner Workflow to accounts in Luminis.

Deploying the CAR File

Note

If the Banner Workflow server is configured to use SSL, and you have configured the Banner Workflow channels to communicate over an SSL port, then the Banner Workflow Web Server’s certificate will need to be imported into the Luminis trusted keystore. You can use the Luminis checkssl tool to do this, for example, 'checkssl myserver 443'. See the Luminis documentation for further details on checkssl.

1. Enable Luminis Integration. Open the configuration.xml file and locate the <LuminisIntegration> element. Change the enabled attribute from false to true. This will instruct the build scripts to generate a workflow.car file for integration.

2. Within WORKFLOW_HOME build and update the workflow.car file by running bin/wftool car. The workflow.car file will be located in the following directory:

WORKFLOW_HOME/dist

3. Deploy the CAR file. Copy the WORKFLOW_HOME/dist/workflow.car file to CP_ROOT/webapps/luminis/WEB-INF/cars.

4. Restart the Luminis Web Service. For the CAR to be available to the channel administrator, the Luminis web service must be restarted. Please consult your Luminis documentation for the proper steps to restart the service.
Configuring Single Sign On from Luminis to Banner Workflow

To allow single sign on from Luminis to Banner Workflow, the configman tool from Luminis must be run to add the connection between the two applications.

In the \$WORKFLOW_HOME/dist directory, a workflowCar.properties file contains the properties that need to be added to Luminis. This file is to be used as an import into the configman utility. To run configman, run `configman -i workflowCar.properties`.

Banner Workflow must be identified as an external system in Luminis using the external system id assigned in the configuration.xml, by default named `bannerwf`. To add this to the names already in your Luminis environment:

1. Check to see what is already present as defined external systems. To do so run `configman -g es.systems`. This will return back the system values that are configured for your Luminis environment.

2. Save this value and use it on our next run of configman.

3. Run `configman -s es.systems <previous value of es.systems> bannerwf`. For example:
   
   if configman -g es.systems returned `library banssb` then you would run `configman -s es.systems "library banssb bannerwf"`.

Configuring Banner Workflow for Luminis SSO

For Single Sign On (SSO) from Luminis to Banner Workflow to work properly, Banner Workflow must be configured to use the Luminis LDAP server for External Authentication. For complete details on enabling external authentication see “Authentication” on page 143.

These instructions assume that you have some knowledge of LDAP and Luminis and are aware of the LDAP configuration settings used during your Luminis installation.
Luminis 4 Configuration

In Luminis 4, the login ID is mutable, and users have an immutable key stored in the uid attribute. The Luminis username is stored in the pdsLoginId attribute, with a possible alternate identifier stored in pdsLoginAlias. So the configuration should use uid for the link.attribute, and a search.filter that looks at both pdsLoginId and pdsLoginAlias.

For example, if your Luminis installation uses the default LDAP server running on a host myserver.myschool.edu at the default port and with all Luminis users stored under ou=People,o=myschool.edu,o=cp, then you would modify the <ExternalAuthentication> element to look like the following:

```xml
<Authentication mode="WorkflowExternal">
  <ExternalAuthentication>
    <ClassName>com.sungardhe.workflow.security.LDAPSearchAuthenticator
    </ClassName>
    <Properties>
      <Property name="java.naming.factory.initial" value="com.sun.jndi.ldap.LdapCtxFactory"/>
      <Property name="java.naming.provider.url" value="ldap://myserver.myschool.edu:389"/>
      <Property name="directory.user" value="uid=wfsearcher,o=myschool.edu,o=cp"/>
      <Property name="directory.user.password" value="password"/>
      <Property name="search.directory" value="ou=People,o=myschool.edu,o=cp"/>
      <Property name="search.filter" value="(|(pdsLogindId=\{0\})(pdsLoginAlias=\{0\}))"/>
      <Property name="link.attribute" value="uid"/>
    </Properties>
  </ExternalAuthentication>
</Authentication>
```

In Banner Workflow, user accounts linked to Luminis users should have their externalID field set to the immutable key stored in the uid attribute in Luminis.

Note

If you have an existing Banner Workflow system integrating with Luminis III, and upgrade to IV, you must edit all the externally authenticated Banner Workflow users and update their externalID fields to match the (possibly new and different) uid attributes of the corresponding accounts in Luminis. You can do this manually, or use the new Web Services exposed with Banner Workflow 8.0 to automate this process.
Configuring Luminis for IDM or CAS Authentication

This section contains the steps needed to integrate Banner Workflow with Luminis. Banner Workflow channels use HTTP/HTTPS to make connections from a channel in Luminis to the Banner Workflow server.

Deploying the CAR File

Note

If the Banner Workflow server is configured to use SSL, and you have configured the Banner Workflow channels to communicate over an SSL port, then the Banner Workflow Web Server's certificate will need to be imported into the Luminis trusted keystore. You can use the Luminis checkssl tool to do this, for example, 'checkssl myserver 443'. See the Luminis documentation for further details on checkssl.

1. Enable Luminis Integration. Open the configuration.xml file and locate the <LuminisIntegration> element. Change the enabled attribute from false to true. This will instruct the build scripts to generate a workflow.car file for integration.

2. Within WORKFLOW_HOME build and update the workflow.car file by running bin/ wftool car. The workflow.car file will be located in the following directory:

   WORKFLOW_HOME/dist

3. Deploy the CAR file. Copy the WORKFLOW_HOME/dist/workflow.car file to CP_ROOT/webapps/luminis/WEB-INF/cars.

4. Restart the Luminis Web Service. For the CAR to be available to the channel administrator, the Luminis web service must be restarted. Please consult your Luminis documentation for the proper steps to restart the service.
Channel Administration

Install the Worklist Channel

The Worklist Channel displays a Banner Workflow User's worklist within the portal format. From the channel, workflow users may interact with, launch, and complete work items. Consistent with the traditional worklist access, as items are completed and when new tasks arise, work items are published and removed from the worklist accordingly. To publish the Worklist Channel:

1. Log onto Luminis with a user who has Channel Admin rights.

2. Click **Channel Admin** within the header of the portal.

3. Click **Publish a New Channel**.

4. Select the *Custom* channel type.

5. Apply the following settings:

   5.1. Define a channel title. This will be the value displayed at the top of the channel. For example:

      *My Worklist*

   5.2. Define the channel name. This will be the name that will be displayed when subscribing to a channel. For example:

      *My Worklist*

   5.3. Define a functional name. This is the functional name of the channel used for identification for JNDI lookups and web services. The channel functional name should uniquely identify this channel definition. For example:

      *worklist.channel.mal0201500_7777_workflow*

   5.4. Set channel timeout. The suggested value is 10000.

   5.5. Set the channel class to

      *com.sungardhe.workflow.luminis.channels.worklist.WorklistChannel.*

6. Set the editable channel control

7. Select the category that will contain the channel. For example:

   *Application*
8. Select the groups who should have access to the channel. For example:

   *Employee*

   **Note**
   
   If a member of this group is not a Banner Workflow user they will be unable to use the Worklist Channel.

9. Click **Finish** to publish the channel.

**Install the Workflow Alert Channel**

Geared towards workflow process owners and administrators, the Workflow Alerts Channel provides portal access to manage workflow instances. If a workflow process should encounter an error state, an alerts message is published to the process owners. The alert provides the owner/administrator with the ability to correct/resolve the process error. To publish the Alert Channel:

1. Log onto Luminis with a user who has Channel Admin rights.

2. Click **Channel Admin** within the header of the portal.

3. Click **Publish a New Channel**.

4. Select the **Custom** channel type.

5. Apply the following settings:

   5.1. Define a channel title. This will be the value displayed at the top of the channel. For example:

      *My Workflow Alerts*

   5.2. Define the channel name. This will be the name that will be displayed when subscribing to a channel. For example:

      *My Workflow Alerts*

   5.3. Define a functional name. This is the functional name of the channel used for identification for JNDI lookups and web services. The channel functional name should uniquely identify this channel definition. For example:

      *alert.channel.mal0201500_7777_workflow*

   5.4. Set channel timeout. Suggested value is 10000

   5.5. Set the channel class to

      *com.sungardhe.workflow.luminis.channels.alert.AlertChannel*. 
6. Set the editable channel control.

7. Select the category that will contain the channel. For example:

   Application

8. Select the groups who should have access to the channel. For example:

   Employee

   **Note**

   If a member of this group is not a Banner Workflow user they will be unable to use the Alert Channel.

9. Click **Finish** to publish the channel.

## Install the Workflow Processes Channel

The Workflow Processes Channel provides Banner Workflow Users with the ability to directly/manually start workflow processes. This is the same functionality that is available via My Processes within Banner Workflow. To publish the Shortcut Channel:

1. Log onto Luminis with a user who has **Channel Admin** rights.

2. Click **Channel Admin** within the header of the portal.

3. Click **Publish a New Channel**.

4. Select the **Custom** channel type.

5. Apply the following settings:

   5.1. Define a channel title. This will be the value displayed at the top of the channel. For example:

       *My Processes*

   5.2. Define the channel name. This will be the name that will be displayed when subscribing to a channel. For example:

       *My Processes*

   5.3. Define a functional name. This is the functional name of the channel used for identification for JNDI lookups and web services. The channel functional name should uniquely identify this channel definition. For example:

       *processes.channel.mal0201500_7777_workflow*

   5.4. Set channel timeout. Suggested value is 10000
5.5. Set the channel class to `com.sungardhe.workflow.luminis.channels.process.ProcessesChannel`.

6. Select the category that will contain the channel. For example:

   Application

7. Select the groups who should have access to the channel. For example:

   Employee

   **Note**

   If a member of this group is not a Banner Workflow user they will be unable to use the My Processes Channel.

8. Click **Finish** to publish the channel.

### Adding a Workflow Tab

Banner Workflow can be embedded into Luminis via a framed tab. For example:

To Set up a Banner Workflow Tab:

1. Logon to Luminis.

2. Select **Content/Layout**.

3. Click **Add New Tab**.
4. Specify a name for the tab. This value will be the displayed value on the tab. For example:

   Workflow

5. If using WorkflowExternal Authentication, select the Framed tab type and supply the CPIP url. The URL field provides the CPIP URL back to Banner Workflow.

   http://<luminis.school.edu>/cp/ip/login?sys=BannerWF&api=workflow

   Where luminis.school.edu is the root of your Luminis server.

   **Note**

   If you are using CAS or IdmGateway as the Authentication mode, simply set the url used to log into your Banner Workflow instance with the relative path after the context root. For example:

   http://workflow.school.edu/workflow/home/worklist.do?renderer=luminis&hidecrumbs=false&hidenav=false

6. Select the desired position of the tab.

7. Click **Submit**.
Section II

Administration
The Server Administration section contains information on the following topics:

- “Server Time Synchronization” on page 87
- “Starting Banner Workflow” on page 87
- “Stopping Banner Workflow” on page 88
- “Import and Export Tools” on page 88
- “Modifying .ZIP Files” on page 92
- “Archive and Purge” on page 94

### Server Time Synchronization

If you are running multiple engine instances, or are using clustering, it is CRITICAL that the system clock on each server be accurate. If the system clock on the servers are out of sync, various pieces of workflow, such as SSO, and execution of automated activities may function erratically, or generate unexpected errors. You should periodically check that the time on the servers is correct, or use a script or some other automated process to periodically check and update the time on each server.

### Starting Banner Workflow

To start Banner Workflow:

1. Start at least one Workflow Engine. To start the default engine, execute `WORKFLOW_HOME/engine/bin/startengine`. See “Deploying and Managing Multiple Engine Instances” on page 43 for details on managing multiple engines.

2. Start the OC4J instance within the Oracle Application Server. To do so, you can use the Enterprise Manager, or the dcmctl tool for Release 2, or opmnctl for Release 3. See your Oracle documentation for further details.
Stopping Banner Workflow

1. Either via Oracle Application Server Enterprise Manager or Oracle command line tools (dcmctl for Release 2, and opmnctl for Release 3).

**Note**
Oracle recommends that only one administrator access the enterprise manager or command line utilities at one time.

2. Stop the workflow engine(s). To stop the engine(s), go to the \WORKFLOW_HOME/\engine/bin directory and execute:

```
engineconsole stop -all -password password
```

Where password is the engine administrative password you selected during installation. ('password' is the default). This will send a shutdown signal to all configured engine instances. See “Scripts” on page 155 for the full command syntax available through the engineconsole command.

Import and Export Tools

Banner Workflow can save and load the contents of the model time data into a zip file that is convenient for backup and migration purposes. This text file is referred to as the import, export, or bootstrap file depending on how it is used.

Import files in Banner Workflow are maintained as a combination of xml documents and binary files contained within a zip file by default. The xml file is based on the schema contained in import.xsd. They can be viewed and manipulated with xml tools, and can also be processed by an XSL.

The Import files contain XML representations of administrative system objects, such as Users, Roles, Workflow Definitions, and Components. They do not contain any runtime information, such as running workflows. Binary documents such as modeler icons are also stored within the zip file.

Under normal circumstances, all import/export processes are carried out from the server console using the installed scripts.

The following command line options are available for importing and exporting:
<table>
<thead>
<tr>
<th>Command Line Option</th>
<th>Description</th>
</tr>
</thead>
</table>
| -primaryKey         | This option applies to both importing and exporting.  
-When importing, it specifies that any primary keys found in the import file should be honored and re-created in the database as-is.  
-When exporting, it specifies that primary keys should be exported with the rest of the data. |
| -verbose            | This option applies to both importing and exporting. The -verbose option will cause the importer/exporter to print verbose error messages. This is mostly used for debugging purposes. |
| -zip                | The -zip option applies only to exports. It is possible to export binary content such as the contents of the global attachments container and since these can be large files, it is not convenient to represent them within the xml content. By default, the xml file will be placed inside a zip file, and other binary content will be exported as binary files within the zip file. The -zip flag is still available for backward compatibility with scripts that use it, although the default behavior is to export to zip unless the -xml option is used. |
Import a File

To import a file, use the following format and command line options:

```
WORKFLOW_HOME/bin/import <username> <password> <import file> [options]
```

For example:

```
import wfroot password import.zip
```

**Note**

When importing a .zip file, the importer will extract the xml file from the zip archive and import it along with the contents of the global attachments container and icons.

**Note**

For the first import, there will be no normal accounts, so you must use the default wfroot account. See “SecurityIntegration” on page 105 for details on how to set the password on the wfroot account.

**Note**

Standard security Authorization checks apply during the import/export processes. If the account used does not have permissions to access an object, the import/export will not be able to create/extract that object. For example, if a user cannot create roles through the UI, the user will also not be able to import roles. Additionally, to import/export, a user must be a member of the ug_admin_remote_services group.
If errors occur during the import process, the system will continue to try to load as many objects as possible. After the file has finished loading, a list of any errors will be displayed. The objects contained in an import file have numerous interrelationships, so that an error preventing one object from loading, can lead to a chain of errors preventing other objects from loading. If errors are encountered, it is best to address them in the order they are listed.

The import process will not change existing objects in the system, it will only create new ones. If you are importing into a database that already contains objects, you may encounter errors if the system tries to import an object with the same name. In this case, you will need to determine whether this is actually a problem or not. If you know that the objects are truly duplicates of one another, then you may safely ignore them. For example, you may be importing a Workflow Definition with its supporting components, roles, and categories into a production database that already contains the same component.

**Note**

If the import file contains an Obsolete Workflow Definition, the import of the definition may fail with validation errors, and may only be able to be imported in the Development state. This is because once a Definition is marked Obsolete, certain validation restrictions are relaxed, and it may become out of sync with the other Banner Workflow objects that it references. When it is re-imported, it may no longer be valid, which may prevent the import.

### Export the Current Banner Workflow System

To export the current Banner Workflow System, use the following format and command line options:

```
WORKFLOW_HOME/bin/export <username> <password> <export file> [options]
```

For example:

```
export wfroot password export.zip
```
Modifying .ZIP Files

If you want to generate your own zip files for import, or edit existing ones, there are several details to keep in mind:

ZIP File Name

The exporter will always create a file ending in a .zip extension, and the xml content will always be placed in a file with a .xml extension in the root of the zip archive. So `export myname mypassword sample.zip` will produce a file named `sample.zip` containing a `sample.xml` file with the xml content. The `staticdocuments` directory will contain binary content, such as global attachments and image icons.

The importer uses the filename suffix to determine whether to import in normal text or zip mode. If the import filename ends with .zip (not case sensitive), then it will assume that the file is a zip archive conforming to the import/export format.

Relationship Between the XML Content File and the staticdocuments

All information on the global attachments (document name, etc.) is contained in the xml file. The binary content is placed in separate files in the `staticdocuments` directory in the zip archive. The files are renamed as file1, file2, etc, to avoid duplicate names. The file names are mapped to their true names within the xml file, and will be re-imported correctly.

Important Notes

- During an import, the objects are created through calls to the application interfaces in the server which controls all transaction boundaries. Essentially, each object is created and committed in a single transaction.

- The order that objects are listed in the xml document is dictated by the xml schema. The import tools know the correct order that all objects must be created in the database to correctly create the relationships between Banner Workflow objects.

- While Banner Workflow does not have tools to selectively export single objects, it does have tools to manipulate a complete export file and extract Workflow Definitions and their supporting objects. For more information, see “Extractor Tool” on page 94.
Additionally, any tools used to manipulate xml documents can be applied to an export file to selectively modify it or extract specific objects into a new import file. The castor framework contained in the package com.sungardhe.workflow.services.importer.castor can also be used to manipulate the import files from java code. See the delivered code samples in WORKFLOW_HOME/examples for an example of this.

- Date ranges are treated as absolutes, and date ranges that fall into the past will be loaded as-is. Additionally, all effective date ranges observe both date and time, not just the date.

**Import File Manipulation with Castor Generated Classes**

If you are familiar with java programming, you can easily manipulate import files or even create them from scratch.

To compile and use the examples, you will need the following in your classpath:

```java
WORKFLOW_HOME/lib/jdom/jdom.jar
WORKFLOW_HOME/lib/jdom/jaxen-core.jar
WORKFLOW_HOME/lib/jdom/jaxen-jdom.jar
WORKFLOW_HOME/lib/jdom/saxpath.jar
WORKFLOW_HOME/lib/xerces/xercesImpl.jar
WORKFLOW_HOME/lib/xerces/xml-apis.jar
WORKFLOW_HOME/lib/workflow/workflow.jar
WORKFLOW_HOME/lib/castor/castor.jar
WORKFLOW_HOME/lib/castor/jakarta-oro.jar
WORKFLOW_HOME/lib/castor/jakarta-regexp.jar
WORKFLOW_HOME/lib/CommonUtil/CommonUtil.jar
WORKFLOW_HOME/lib/log4j/log4j.jar
```

For the purposes of the example, assume that you have a flat file containing a list of usernames that we want to add to an import file before loading it into the system. All of the new users will be given a default password of `changeme` and the role `Clerk` as a primary assignment. Clerk is a role defined in the original import file.

See the `WORKFLOW_HOME/examples/castor/src/demo/NewUsers.java` file for a commented example of how to add a new user. Once compiled, the example will either add new users to an existing import file if `-source` is specified or will create a new file containing only new users if `-source` is omitted.
Extractor Tool

Banner Workflow ships with an extraction tool that can be run against import/export xml files to extract a Workflow Definition, along with the roles, components, categories, product types, and technology types that support the workflow.

To use the extractor tool, run the extractwd script as follows:

```
WORKFLOW_HOME/bin/extractwd [-shell] | { [-withoutDependencies] -source <sourcefile> -target <targetfile> -processdef {<organization><name> <version>}* }
```

If the [-withoutDependencies] flag is specified, only Workflow Definitions will be exported. Otherwise, the definition and supporting object such as roles, components, and categories will also be exported.

If both the source and target file for the extractor have a .zip suffix and [-withoutDependencies] is not specified, any global attachments referenced by the extracted workflows will also be placed in the target file. The <organization> refers to the qualified organization name of the workflow definition.

Since this process only involves xml manipulation, it is not necessary to have the workflow server running. You can also use the -shell option to open an interactive shell to enter the information. For example

```
WORKFLOW_HOME/bin/extractwd -shell
```

In shell mode, you will be successively prompted for the target, source, and process definition to extract.

Archive and Purge

Archive Tool

The archive process enables you to write information about stopped and completed workflows to an Oracle repository. Queries can then be run against the repository for report generation. See “Archive” on page 3 for a diagram outlining the archive tables and their relationships. Archived workflows are purged from the main system once they have been successfully archived.

**Note**

It may be necessary to use -shell option on certain operating systems (typically Unix variants) if there are spaces in either the qualified organization name or the definition name.
Parameters

When running the archive process, data is written to the archive based upon the values assigned to each of the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>user</td>
<td>Identifies the user name. This must be a valid workflow user ID.</td>
</tr>
<tr>
<td>password</td>
<td>Specifies the password for the user.</td>
</tr>
<tr>
<td>verbose</td>
<td>Generate verbose output</td>
</tr>
<tr>
<td></td>
<td>• True: verbose output will be written to the standard output.</td>
</tr>
<tr>
<td></td>
<td>• False: basic output will be written to the standard output.</td>
</tr>
<tr>
<td>displayContext</td>
<td>Show context parameter values.</td>
</tr>
<tr>
<td></td>
<td>• True: values of workflow context parameters will be written to the archive.</td>
</tr>
<tr>
<td></td>
<td>• False: values of workflow context parameters will not be written to the archive.</td>
</tr>
<tr>
<td>days expired</td>
<td>Minimum number of days a workflow must be completed or stopped for, to be included in the archive.</td>
</tr>
</tbody>
</table>

You can also pass an optional `-batchSize <size>` option to control the size of the batches in which workflows will be archived. By default, the batch size is 50.

Archive Format

To archive, use the following format and command line options:

```
archive 'user' 'password' 'verbose' 'display context' 'days expired'
```

The following command will archive all workflows that have been stopped or completed for 20 or more days. Verbose output will be written to the output stream and the values of context parameters within each archived workflow will be written to the archive:

```
archive admin password true true 20
```
Purge Tool

The purge process enables you to remove permanently stopped or running workflows. A significant difference between the purge and archive tools is that the purge leaves no memento of the original workflow object.

Use the purgewf script to purge with the following format:

```
WORKFLOW_HOME/bin/purgewf <username> <password>
{-shell | {filter terms}} [-noprompt]
```

Where 'filter terms' may consist of:

- `-workflowOrg <workflow organization qualified name>`
- `-processName <business process name>`
- `-definitionName <workflow name>`
- `-definitionVersion <version number>`
- `-instanceName <instance or specifics name>`
- `-state <state>`
- `-startDateFrom <date>`
- `-startDateTo <date>`
- `-endDateFrom <date>`
- `-endDateTo <date>`

**Note**

*<state> has a default value of stopped*.

<date> must be formatted in one of the following accepted date formats:

- `dd-MMM-yyyy` (e.g., 20-JAN-2006),
- `dd-MMM-yyyy hh:mm:ss a` (e.g., 20-JAN-2006 01:00:00 PM),
- `dd-MMM-yyyy'T'HH:mm:ss` (e.g., 20-JAN-2006T13:00:00).

Example: The sequence below would remove all stopped workflow from the 'Root' organization that completed by October 13, 2006.

```
purgewf admin password -workflowOrg Root -endDateTo 13-OCT-2006T17:00:00
```

The script will prompt if it should carry out the workflow purge unless the –noprompt is passed into it.

You may use the –shell option to interactively enter any or all of the filter terms instead of passing them in all at once at the command line.
Example:

    purgewf admin password -shell

If there are spaces in any of command line arguments it may be necessary to use the -shell option.

**Note**

There is currently a limitation on the number of command line arguments that can be passed via purgewf script to the underlying java executable. Use the –shell option if specifying many different filter terms.
The Database and Server Administration section contains information on the following topics:

- “Redeploying” on page 99
- “configuration.xml” on page 100
- “Manipulating Data Sources through Banner Workflow” on page 111
- “Connection Management” on page 112
- “Automated Activity Deployment” on page 114
- “Clone a Banner Workflow Database” on page 115

Redeploying

Some changes made to the configuration.xml require the various workflow artifacts (the EAR file, the car file, the engine nodes) to be rebuilt/redeployed in order to take affect.

You can rebuild all the workflow artifacts, and upload configuration changes to the database in a single step by using wftool with the updateSystem command:

```
WORKFLOW_HOME/bin/wftool updateSystem
```

After updateSystem runs, you will have a new EAR file ready for deployment to the Application Server. If Luminis configuration is enabled, a new CAR file will be ready for deployment to the Luminis server. The default engine node will also have been rebuilt and a new engine installer will have been created for installing separate engine nodes.

**Note**

The only configuration change that requires you to re-install engine nodes is if you change the Banner Workflow datasource url, username, or password. Any other configuration changes will automatically be picked up by the engines after they are restarted.
**configuration.xml**

Configuration.xml is the master configuration file, controlling how Banner Workflow is deployed on the ApplicationServer and how it behaves at runtime. This file is loaded into the database during deployment. Sections not related to deployment can be modified after deployment and reloaded into the database, so that changes can be made without requiring redeployment.

To make changes to the configuration.xml file:

1. Edit the file, or use the configwizard to change the file.
2. Run WORKFLOW_HOME/bin/wftool uploadconfig to reload the file into the database.
3. Use the engineconsole and startengine scripts to restart the engine node(s).
4. Restart the OC4J instance(s).

**Note**

If you change settings under the `<Deployment>` element, you will generally have to rebuild and redeploy the ear file for these settings to take effect. In the descriptions that follow, the elements will specify if a redeployment is necessary after an element is changed.

The configuration file contains the following elements:

- “EmailServer” on page 100
- “AutomatedActivityServlet” on page 102
- “Reaper” on page 102
- “EventDispatcher” on page 102
- “DataSources” on page 103
- “SecurityIntegration” on page 105
- “Engines” on page 106

**EmailServer**

Defines properties (such as the smtp mail host) that are used to send email for Email activities. If you are using email activities, you should adjust these settings to point to your mail host.

Changes to the EmailServer element require both the OC4J instance and the engine to be restarted to take effect.
Properties

Used to configure Banner Workflow to talk to your email server.

The required property is `mail.smtp.host` which specifies the location of the email server. The name of each property is specified in the JavaMail API. For additional configurations please refer to Sun's documentation concerning the API.

SubstitutionProperties

Substitution properties are values that are available to the Email Service when formatting an email.

Substitution properties can be added for global email substitutions such as default Workflow From addresses and standard footers.

For example if you’d like to have a default footer be placed at the bottom of each email being sent via Banner Workflow and you would like this footer to be the same throughout the system you could create a substitution parameter named “FOOTER” with a value of “brought to you by Banner Workflow” in the configuration.xml.

```xml
<EmailServer>
  <Properties>
    <Property name="mail.smtp.host" value="localhost"/>
    <Property name="mail.smtp.sendpartial" value="true"/>
    <Property name="mail.smtp.dsn.notify" value="FAILURE"/>
    <Property name="mail.smtp.dsn.ret" value="FULL"/>
  </Properties>
  <SubstitutionProperties>
    <Property name="FOOTER" value="brought to you by Banner Workflow"/>
  </SubstitutionProperties>
</EmailServer>
```

At the bottom of the email you’d place `@FOOTER` to bring in the value of FOOTER from your configuration.xml file. By keeping the value in one place you’re able to change once and have the values picked up automatically by future email activities.
AutomatedActivityServlet

The maxThreads attribute specifies the maximum number of threads that will be started to concurrently perform automated work.

The pollingInterval specifies the time, in seconds, to pause between checking for new automated work.

The continousPolling attribute instructs the automated activity servlet to ignore the pause interval if the last poll found work to be performed. Caution should be exercised before setting this attribute to true. While doing so may increase automated activity throughput, it may do so at the expense of system response to user interaction.

Note
Changes to AutomatedActivityServlet require the OC4J instance to be restarted.

Reaper

The PurgeInterval element defines how often the reaper threads should purge expired Banner SSO and logon data. Normally this data is purged during program execution, and the reaper threads act as a backup to this functionality. The time is displayed in seconds.

Note
Changes to Reaper require the OC4J instance to be restarted.

EventDispatcher

Defines the event providers. Banner Workflow can respond to events from multiple providers. Each <EventProvider> element names a provider, specifies the java class that implements that provider, and specifies whether the provider should be enabled or not. Each <EventProvider> element contains a <Properties> element specifying name-value pairs used to configure the provider. The name-value pairs are specific to each provider class. The <EventProvider> element also has an “enabled” attribute that can be used to switch the provider on or off (set it to “true” for on, and “false” for off).

Note
Changes to the EventDispatcher require the engine(s) to be restarted.

Banner Workflow ships with a BannerTableExternalEventProvider. The name-value pairs used to configure this provider are as follows:
sourceID Specifies the source of the External Events handled by this provider. All External Events posted to workflow are required to have a source ID that uniquely identifies the external system generating the events. (See “External Events” on page 139 for information on source IDs.)

pollingInterval The delay (in seconds) between attempts to poll the Banner event tables.

dbcDriver The name of the jdbc driver to use.

connectionUrl The jdbc url to the database containing the Banner event tables.

user The username to use to logon to the database.

password The password to use to logon to the database.

maxResultSizePerPoll The maximum number of events to retrieve per poll.

targetName The event target name; should always be WORKFLOW. Identifies the events in the Banner Event tables that should be polled by this provider.

name The name the provider should use when generating log messages. In general, this should be the same as the provider name.

DataSources

This element contains zero or more defined datasources. A datasource is a named jdbc connection that stores the information required to connect to an external data store. DataSources defined here can be attached to ProductTypes to provide access to data associated with applications (such as Banner) that your Banner Workflow installation has been integrated with, or needs to acquire data from. A <DataSource> element contains a name attribute, which should be the unique for each DataSource. The <Url>, <Username>, and <Password> elements define the jdbc Url and connection information.

Note

Changes to any of the DataSource elements require OC4J to be restarted.
It should be noted that the DataSource elements defined here are not the same as the WorkflowDataSource and BannerDataSource elements defined in the Deployment section. These datasources are used to provide JDBC connection information to workflow activities, such as automated sql stored procedures, as such, they will often use different database accounts to login to, with different privileges than the accounts used in the Deployment datasources.

Each datasource contain the following elements:

- **Url**
  JDBC connection URL, for example:
  `jdbc:oracle:thin:@sunhost1:1521:BTest`

- **Username**
  ID used by Banner Workflow to logon to a database instance.

- **Password**
  Password used by Banner Workflow to logon to a database instance

### For Example

```xml
<DataSources>
  <DataSource name="BTest">
    <URL>jdbc:oracle:thin:@sunhost1:1521:BTest</URL>
    <UserName>user</UserName>
    <Password>password</Password>
  </DataSource>
</DataSources>
```

**Note**

If the Data Source does not require logon information leave the text between the UserName and Password tags blank. If the Data Source requires a logon, ensure that the UserName/Password combination used has been granted privileges to all objects accessed by the Banner Workflow system.
What types of external data sources can be defined

Banner Workflow can access two JDBC drivers, an oracle driver, and an ODBC bridge driver. This allows Banner Workflow to interface with Oracle databases and system defined ODBC data sources, such as, Microsoft Access, Microsoft FoxPro, or Microsoft Excel. The Connection URLs for Data Sources accessed via these drivers should use the following templates:

- **Oracle Connection URL**: jdbc:oracle:thin:@<hostname>:<port>:<sid>
- **ODBC Bridge Connection URL**: jdbc:odbc:<data-source-name>

Note
Banner Workflow can be configured to access any data store for which your institution has the appropriate JDBC driver.

For more information on using DataSources, see “Manipulating Data Sources through Banner Workflow” on page 111.

SecurityIntegration

This element controls several built-in accounts and determines the authentication mechanism employed by workflow.

Note
Changes to this element require the OC4J instance to be restarted.

<SuperUser>

Controls configuration of the built-in wfroot account. The password attribute determines the password used for the wfroot account, and should be changed from the default of “password”. The <Groups> element should only be changed when requested by SunGard Higher Education.

<WebServicesUser>

Controls configuration of the built-in web service account “wfwebservices”. This is an internal account used for server-to-server integration processes, such as Luminis channels. You should change the password attribute to something other than the default of “password”. <Groups> defines the security groups the account belongs to, and should only be changed when requested by SunGard Higher Education.
<ExternalAuthentication>

Allows an external authentication mechanism, such as LDAP, to be enabled and configured. Enabling external authentication requires that a specific implementation be specified by the <ClassName> element. The <Properties> element then specifies a set of configuration properties specific to the <ClassName> chosen. For specific details on what external authentication mechanisms are supported and how to configure them, see “Authentication” on page 143.

Engines

The Engines element contains all configuration information relating to the Workflow Engine.

Note
Changes made to the Engines element require that the OC4J instance(s) and the engine(s) be restarted.

Overview

The information in the <Engines> element serves two purposes. First, it controls the behavior of the engine instance(s) that are running. Secondly, it allows other Banner Workflow components to find where the engine(s) are located and make RMI calls to them.

Each <EngineInstance> element specifies a location where an engine instance is expected to be running. The host attribute identifies the server, and the config attribute specifies the configuration that the engine is running under. Together, the information denotes both the server and the RMI port that the engine can be contacted at. For typical single-engine installations in which the engine is running on the same server as the OC4J instance, the host will be “localhost” and the config will be the default “main” config.

Note
If you are using application server clustering, you should change 'localhost' to the fully-qualified name of the server holding the workflow installation, even if you are running only one engine. The other nodes in the cluster will need the fully-qualified hostname in order to contact the engine instance.
The `<EngineConfiguration>` elements define one or more configurations that an engine can run under. In a typical single-engine installation, there will only be a single “main” configuration.

The `<EngineConfiguration>` element contains the following information:

- The `<Port>` element specifies the RMI port the engine runs on. This port should not be used by any other application on the server the engine runs on.
- `<JDBCPoolConfig>` controls the maximum number of JDBC connections that the engine can open at once to service incoming RMI requests. For more information see “Tuning the Workflow Engine” on page 165.
- The `<ThreadPools>` element specifies tuning information for the Main, Notification, and ExternalEvent thread pools. For more information see “Tuning the Workflow Engine” on page 165.
- `<LoggingConfig>` specifies a named logging configuration to use for this engine configuration. Logging configurations are sets of name/value pairs that control the level of logging detail.
- `<Password>` specifies the administration password used by all engine instances. Scripts that perform administrative functions on the engine (such as a shutdown request) must provide this password to the engine, or the request will be denied.
- `<MaxMemory>` controls the maximum amount of memory (in megabytes) the engine scripts will automatically be configured to start the engine VM with.
- `<RefreshSettings>` controls how far the engine “looks ahead” for scheduled events. This value should only be changed when requested by SunGard Higher Education.
- `<Services>` lists the RMI services published by the engine, and should not be changed.
- `<DBNoWaitFailureCodes>` and `<DBFailureCodes>` configure the engine to detect database failures so it can attempt to go into an idle state and auto-recover when database connectivity is restored. These values should be altered only when requested by SunGard Higher Education.

**Deployment**

The Deployment section contains information that is related to the building and deployment of the workflow artifacts, such as the EAR file. Information contained here is generally more static, and, unless otherwise noted, requires redeployment in order to take affect.
**WorkflowDataSource**

This element specifies the JDBC connection information and OC4J connection pool tuning parameters for the connection to the Banner Workflow tables.

<JDBCConnection> Specifies the basic JDBC connection settings.

- <Url> - The url to the Banner Workflow database in the form jdbc:oracle:thin:@<servername>:<listener port>:<sid or service>.
- <Username> - The username to use to connect.
- <Password> - The password to use to connect.
- <Driver> - The jdbc driver to use. This should not be changed from oracle.jdbc.driver.OracleDriver.

<max-connections> The maximum number of connections the Application Server can have open in the workflow connections pool. This number controls the number of users that can concurrently interact with most workflow functionality.

<min-connections> The minimum number of connections the Application Server should keep ready to service user requests.

Note

The rest of the elements control JNDI binding of the connection pool, and should not be modified except when requested by SunGard Higher Education.

**BannerDataSource**

This element specifies the JDBC connection information and OC4J connection pool tuning parameters for the connections to the WFBANNER account used to exchange data with Banner when executing Banner workitems.

Note

The rest of the elements control JNDI binding of the connection pool, and should not be modified except when requested by SunGard Higher Education.

**ApplicationName**

This specifies the application name for a given workflow instance. The name specified here will be used to name the Workflow Engine service when running under windows (if this feature is used).
**ApplicationServerHost**

Specifies the hostname the application server is running on, or the hostname of one of the hosts a clustered environment runs on. Will be used when creating utility scripts (import, export, etc).

**WebApplication**

This element specifies the elements of the URL by which users and web clients can connect to the application. If you are using multiple Oracle http servers behind a load balancer, this should specify the URL of Banner Workflow as accessed through the load balancer.

- **protocol**
  Either http, or, if you are using SSL, https.

- **host**
  The name of the server the Oracle Application HTTP server is running on (typically the same server Banner Workflow is installed on). It should be a name resolvable by DNS by any potential web client (browser or Luminis channels). It should not be ‘localhost’.

- **port**
  The port that the HTTP server is running on.

- **root**
  The web root of the deployed application. Root should denote the type of system being installed. Typically this should match or be similar to `<ApplicationName>` for convenience. For example, if deploying a production system, you might use `root="/wfProd"`.

**Note**

The forward slash at the beginning of the name is required.

**LuminisIntegration**

This element is used when integrating Banner Workflow with Luminis. To enable integration with Luminis, the enabled flag must be set to true.

- `<ExternalSystemID>`
  Contains the external system to be added to the es.systems configuration value in Luminis (via the Luminis configman tool) to specify the CPIP integration from Luminis to Banner Workflow.
AdvancedDeployment

This element contains a number of advanced deployment features.

<LoggingConfig> Specifies a named logging configuration to use to control logging from all Banner Workflow components running in OC4J.

<ScriptExtensions> Controls the extensions used on scripts created by the wftool. If you wish the scripts to be created with different extensions, change this element, then use “WORKFLOW_HOME/bin/wftool scripts” to recreate the scripts.

Note
Changes to this element do not require a redeploy, just recreate the scripts using the wftool.

<WebSessionTimeout> The time (in minutes) before an idle web session will be automatically logged out.

<JavaWebStart> Contains parameters used to control the launching of parts of workflow that use Java Web Start for rich-client functionality, such as the modeler. The <MaxMemory> element controls the amount of memory (in megabytes) that the client side VM will be allowed to use. The <J2SE> element controls the version(s) of JDK that are acceptable to run the workflow UIs, and should only be changed at the request of SunGard Higher Education.

Logging

The <Logging> element specifies a number of named logging configurations. Each configuration is contained in a <Log4j> element having a name attribute. Each name must be unique. The contents of a <Log4j> element is a <Properties> element specifying the name/value pairs used to configure log4. You can change the detail of the logs generated by altering the logging configurations. By default, all the workflow components running in OC4J are controlled by the 'appserver' logging configuration, and all the engine nodes are controlled by the 'engine' logging configuration.
Manipulating Data Sources through Banner Workflow

Banner Workflow provides two internal Java programs to manipulate data in an external data store. The first program executes an external Data Store's stored procedures. The second program allows you to execute SELECT statements against an external Data Store. Before these two programs can be used as part of a component that manipulates the external Data Store, three steps must be completed:

1. A DataSource element specifying how to connect to the data store must be defined in the configuration.xml file, located in the config directory of the Banner Workflow installation directory.

2. A new product type must be created and configured to use the newly defined DataSource.

3. A new technology type must be created with a client launch parameter named “jdbcDriver” with a value equal to the classname of the JDBC driver needed to connect to the external data store. The paths for the two drivers Banner Workflow has access to are:

   - Oracle Driver Class: `oracle.jdbc.driver.OracleDriver`
   - ODBC Bridge Driver Class: `sun.jdbc.odbc.JdbcOdbcDriver`

Once the product type and the technology type have been defined, a component can be created to execute a stored procedure or a SQL select query. Detailed information on creating such a component is contained in the Analyst/Administrator Handbook.

In the case of a MIF environment, a MIF code must be specified for each relevant organization that a workflow may be instantiated. See the Banner Workflow Analyst/Administrator Handbook for more information regarding MIF.
Connection Management

Banner Workflow makes use of connection pooling to better utilize database resources. Proper tuning of the connection pools requires an understanding of where and how connections are used.

Two connection pools are maintained by the application server:

1. **Workflow Connection Pool** - The pool defined in configuration.xml under WorkflowDataSource. This can be changed by editing the `<max-connections>` element and redeploying. These connections are used for all access to the tables in the workflow schema from the web and ejb tiers, including reading workflow data from the eng * tables.

2. **Integration Connection Pool** - The integration connection pool is used to integrate with Banner. These connections are used to pass integration information during workflow launch and for getting and setting parameters from Banner. This has to be a separate connection pool, as Banner Workflow and Banner will most likely reside in different databases and would require different credentials to log in. This connection pool is defined in the Banner Data Source element in configuration.xml.

Changes to either of the above elements requires the EAR to be rebuilt and redeployed.

The Workflow Engines run in separate VMs and maintain their own connections to the database. All connections from the engines use the JDBC connection information contained in the `<WorkflowDataSource>` element. The maximum number of connections that can be used by an engine instance is the sum of maximum size of its JDBC pool, the maximum number of main pool threads, the maximum number of notification pool threads, the maximum number of external event pool threads, plus 1. For complete details on how the engine makes use of database connections, see “Tuning the Workflow Engine” on page 165.

The provided BannerTableExternalEventProvider that transfers events from Banner to Banner Workflow opens one connection to the Banner Database per instance of the provider. Each engine instance that is running will have its own instance of the provider running. So if you are running 2 engine nodes and are polling for Banner events, there will be 2 JDBC connections to the Banner event tables.
Automated activities that access databases may also use database connections. These connections are opened and closed by the automated activities themselves, and do not come from any pools. The maximum number of external activities that could concurrently exist can be controlled by editing the `<AutomatedActivityServlet>` element of the configuration.xml file and adjusting the max external processes element. Note that if clustering is being used, each OC4J instance running the Banner Workflow components will have an instance of the automated activity servlet running. So the total number of connections that could be in use by automated activities is the maximum number of external processes allowed by the automated activity servlet multiplied by the number of OC4J instances running the Banner Workflow system.

Tuning the connection pools depends both on the typical operating mode of Banner Workflow and your Oracle connection licensing.

Banner Workflow operating modes vary between highly automated systems that mostly perform automated activities with low numbers of interactive users at one extreme, to highly interactive systems in which most activities are performed by high numbers of concurrent users.

Oracle licensing models vary from the concurrent user model to the processor based model. With processor based licensing, controlling the pool max sizes is not as critical, since they can be set higher than needed in anticipation of sudden peak loads. The concurrent user model when combined with highly interactive systems presents a challenge in determining how best to allocate the limited resources.

For an installation with a small user base of less than 30 concurrent users or a mostly automated installation, the default settings should be sufficient. If you are on a processor based model, you may wish to increase the size of all pools in anticipation of future demand.

The following are guidelines on which pools to change based on the performance of the system:

- If normal administrative actions involving components, users, roles, or workflow definitions seem slow, and you have an interactive system, try increasing the `<WorkflowDataSource>` pool max size.
- If worklist refresh times or workflow status searches seem slow, try increasing the `<WorkflowDataSource>` pool max size.
- If the Banner response time to a work item launch or a work item complete within Banner seems slow, try increasing the `<BannerDataSource>` max pool size. You may also need to increase the `<JDBCPOOLConfig>` max size for the engine configuration under the `<Engines>` element.
- If interacting with work items seems slow when reserving, releasing, completing, or launching, try increasing the `<JDBCPOOLConfig>` max size for the engine configuration under the `<Engines>` element.
If workflows seem to take a long time to advance after being started or after a workitem is completed, you may need to increase the number of threads in the main engine pool. For more information see “Tuning the Workflow Engine” on page 165.

Note

Production environments should always be running workflows in the 'Active' state (not 'Test'). Running large numbers of 'Test' mode workflows will consume significantly higher database resources in terms of DB reads required.

The other thing to keep in mind with connection pool tuning is that as a distributed application, Banner Workflow is dependent upon network and database performance and maintenance. The connection pool sizes essentially determine the maximum number of concurrent user requests the system is capable of processing without beginning to make users wait for database resources to become available. Slow network or database response will increase the length of each request, driving up the concurrency, and therefore increasing the total number of connections that may be needed.

Automated Activity Deployment

There are three different types of automated activities that can be designated to a component: internal, automated workflow-aware, and automated non-workflow-aware.

Internal

The internal type is reserved for Banner Workflow built-in activities such as the automated SQL procedure and query activities. The binary code of these components is packaged with Banner Workflow and does not have any special deployment considerations. To properly set up these kinds of components you must ensure that the product type and data sources have been properly defined.

In the case of a MIF environment, a MIF code must be specified for each relevant organization that a workflow may be instantiated. See the Banner Workflow Analyst/Administrator Handbook for more information regarding MIF.
Automated Workflow-aware and Non-workflow-aware

The automated workflow-aware and non-workflow-aware activities provide a means for hooking external applications as automated activities in the system. The key difference is that non-workflow-aware activities are automatically marked as completed by the workflow server after the process has completed. Workflow-aware activities are required to communicate with the Banner Workflow server (via Web Services) to complete themselves. In both cases, the automated activities are run in an external process on the same machine as the workflow server. This is analogous to how cgi programs are launched on behalf of a web server.

These types of automated activities should be mapped to a drive that is accessible from the Banner Workflow server. This involves deploying the actual scripts or binary code in an accessible directory, and defining the executable and any required path information in the client launch parameters of the component. The administrator and analyst should take care that the launch executable and path to the physical application file are in sync. Refer to the Analyst/Administrator Handbook for a detailed description in defining the launch parameters with external automated activities. In a clustered environment, any of the OC4J instances running the Banner Workflow system may perform an automated activity, so the executable used by the component must be accessible from each OC4J instance.

Clone a Banner Workflow Database

Occasionally you may want to roll a production instance of Banner Workflow back to a test instance. To clone a Banner Workflow Database, please follow the steps below:

Note
In the following steps the source database is the database we are cloning from and the target database is the database we are cloning to.

1. To ensure that data will not be manipulated during the export, it is important to shut down the Banner Workflow environment. To shut down the Banner Workflow environment, stop the OC4J instance and the Workflow engine of the source Banner Workflow environment.

2. Take a database export of the source database. If a full export is not needed, an export of just the WORKFLOW schema will be sufficient. This schema is listed in the configuration.xml file as the WorkflowDataSource.

3. Upon completion of the export, restart the source Banner Workflow environment.
4. If you want to keep any of the existing work in the target Banner Workflow environment, a Banner Workflow export should be performed. To run an export, please review “Export the Current Banner Workflow System” on page 91. Here is an example on how to export:

\[\text{WORKFLOW\_HOME/bin/export wfroot <password> WorkflowTargetExport.zip}\]

5. Log onto the target's Banner Workflow environment as a user with admin rights. Navigate to the Technology Types page under Administration and Advanced Settings. Click on the Banner Forms technology type to view the settings in this technology type. Take a screen capture or print this screen so it will be available for use in a future step to reset the technology type.


7. Roll the database export from step 2 into the target database.

8. From the target Banner Workflow Environment's WORKFLOW_HOME directory run "bin/wftool uploadconfig" to take the configuration.xml file that exists in WORKFLOW_HOME/config and push it up to the database.

\[\text{Note}\]

This is very important because prior to this step the configuration that is in the database is the configuration for the source environment.

9. If desired you can overlay the export of your target environment on top of the newly cloned database. To do so please refer to “Import a File” on page 90. Here is an example on how to import:

\[\text{WORKFLOW\_HOME/bin/import wfroot <password> WorkflowTargetExport.zip}\]

\[\text{Note}\]

In this step, errors are expected because there will be name collisions between the source system's data and the target system's data. Please review the errors that are provided to ensure that the expected workflow data is imported.

10. Start the target Banner Workflow environment.

11. Log on to the target Banner Workflow environment and go back to the Banner Forms technology type. Select the launch parameters link and ensure that all values are identical to the values in the screen capture or print out from step 5.
6 Client Administration

The Client Administration section contains information on the following topics:

- “Launch Services” on page 117
- “Desktop Application Deployments” on page 120
- “Java Web Start / Certificates” on page 120
- “Desktop.set” on page 122
- “Internationalization” on page 123

Launch Services

A launch service determines how an interactive business component is started from the Worklist or Workflow Status page when associated with an activity in a workflow.

A business component can refer to a form, URL, rich client, or a service. Each type of component has its own logic code to determine how to start the component. A technology type is used to specify which launch service logic and is a required attribute for a business component. In addition, the technology type may define launch parameters that effect the start of the component. These launch parameters are shared by all components with that type.

Launch Services are categorized into either client or web, depending on where the actual launching occurs. In some cases, both a client and a web launcher may be defined for a specific component type. The workflow launcher checks the user’s preference when determining whether to pass control to either the web or client launch service.

Below are the bundled launch services that may be added to technology types in the Business Component Catalog:

<table>
<thead>
<tr>
<th>Launch Service Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.sungardhe.workflow.launcher.SimpleClientLauncher</td>
<td>Client</td>
<td>Opens a generic desktop application.</td>
</tr>
</tbody>
</table>
All of these listed services are for interactive business components, and each is implemented for a specific type of rich client application or web page launch.

**SimpleClientLauncher**

The SimpleClientLauncher requires the launch parameter `executable` to be defined. Unlike automated activities, the `executable` parameter will not be treated as the identical name of the application to launch, but rather a virtual name. At runtime, the user may map this executable name to a physical program accessible from their desktop. Additionally, SimpleClientLauncher may take an unlimited number of optional parameters in the form: `arg0, arg1, arg2, … argN`, where the number after the `arg` prefix is used to mark the order that parameters are appended when forming the single executable string.

**SimpleWebLauncher**

The SimpleWebLauncher requires the launch parameter `host` to be defined, where the value for `host` is a valid URL. The URL is launched using an HTTP POST or GET and submits any launch parameters defined as part of the post data. SimpleWebLauncher defaults to using POST but this may be overridden by specifying a method launch parameter and setting the value to either GET or POST.

**WorkflowawareClientLauncher**

The WorkflowawareClientLauncher is defined the same way as the SimpleClientLauncher. The only difference is that the workflow application will not prompt the user to designate if the work item has been completed or not. It is the responsibility of the launched executable to handle workflow handshaking.

**Banner Launchers**

The Banner related launcher is described in the Banner Integration section of this handbook. See “Banner Integration” on page 47 for more information.
Built-in Launch Parameters

When building client or web launch parameters for a component, you can use the following built-in launch parameters to substitute runtime values into the launch:

- **@BUILTIN_DB_URL**: Substitutes the URL from the data source associated with the component’s product type.
- **@BUILTIN_DB_USERNAME**: Substitutes the user name from the data source associated with the component’s product type.
- **@BUILTIN_DB_PASSWORD**: Substitutes the password from the data source associated with the component’s product type.
- **@BUILTIN_WORKITEM**: Substitutes the value of the current work item key.
- **@BUILTIN_ORG_QUALIFIED_NAME**: Substitutes the qualified name of the organization for the current workflow instance.
- **@BUILTIN_ORG_NAME**: Substitutes the short name of the organization for the current workflow instance.
- **@BUILTIN_ORG_MIF_CODE**: Substitutes the associated MIF code if defined of the organization for the current workflow instance.
- **@BUILTIN_ORG_ID**: Substitutes the database primary key of the organization for the current workflow instance.

Additionally, any component parameter name can be used to substitute a runtime value into a launch parameter. For example:

```
id=@student_id
```

In this example, ID is a launch parameter and Student ID is a component parameter that will be substituted into the ID value. Two examples of using parameterized substitutions can be found in `WORKFLOW_HOME/examples/SimpleClientLauncher` and `WORKFLOW_HOME/examples/SimpleWebLauncher`.
Desktop Application Deployments

It is important to ensure that the client machine has access to any desktop application that it might be prompted to launch from a work item.

Java Web Start / Certificates

Banner Workflow uses Java Web Start technology to extend the capabilities of the browser in the following areas:

- workflow graphical modeler
- desktop client launcher
- desktop application mapping

Java Web Start allows client-side java programs to be launched from a web application and run in Sun's Java Runtime Environment. Unlike applets, the launched programs run in their own client-side VM, and thus avoid JVM collisions with other applications (such as Banner) that rely on applets.

The jars for the client applications are initially downloaded the first time they are used; and may take some time depending on the speed of the network. Once downloaded, they are cached locally and subsequently load much faster. The client-side applications in Banner Workflow support version 5 of the Java Runtime Environment.

Signed Jars

SunGard Higher Education has signed each jar used by the client applications with a digital certificate that guarantees that the client code can not be maliciously modified. A digital certificate is used to prove the identity of the signer.

The SunGard Higher Education client applications require the user's permission to provide the following features:

- Starting work items that launch a program for the user.
- Storing the paths to executables used to launch client applications associated with components in the desktop.set file.
**Note**

If you are using OpenSSL or a similar tool to act as your own certificate authority, you may need to register your SSL root certificate into the java keystore (typically the file `{java home}/lib/security/cacerts`). Both the java jre and java jdk include a utility called keytool for importing a private certificate into the keystore. Here is an example:

```
{java home}/bin/keytool -import -keystore {java home}/lib/security/cacerts -alias my_certificate -file {certificate file}
```

This is usually not an issue for certificates issued by public certificate authorities (such as VeriSign) since their root certificates are pre-installed with java. Please contact the ActionLine if you have any questions.

---

**Java Web Start with Internet Explorer**

When using Internet Explorer with Banner Workflow that has SSL (https) enabled, a Save file as window for JNLP may appear when launching the modeler. If the Save file as window appears:

1. Select Tools>Internet Options.
2. Select the Advanced tab.
3. Under Security, deselect **Do not save encrypted pages to disk**.

---

**Java Web Start with Netscape Navigator**

To enable Java Web Start on the Netscape Navigator browser you may need to complete the following steps:

1. In Netscape, select Edit>Preferences.
2. Click **Helper Applications** under the Navigator heading, and ensure that `application/x-java-jnlp-file` is listed under File Types and is set to **Open these files using the default application**.
3. If there is no entry for `application/x-java-jnlp-file` in the File Types list, click **New Type**.
4. For **MIME Type**, enter `application/x-java-jnlp-file`.
5. For **Description**, enter `jnlp`.
6. For **Extension**, enter `jnlp`.
7. For **When a file of this type is encountered**, select **Open it using the default application**. If the application line is present but not using default, edit the file type and set it to default.

8. Click **OK**.

9. Click **OK**.

10. Close your browser, open a new browser, logon to Banner Workflow, and then launch the Workflow Modeler.

## Desktop.sct

Each component in the Business Component Catalog that launches a client application contains special client launch parameters. The parameters are hints for what application to launch and how to launch it. The parameters are usually specific to the launch service that is invoked. For example, a desktop application requires a client launch parameter named `executable`.

When you launch a work item locally, the launcher needs to run the executable that is associated with the work item's component. To do this, the launcher needs the path to the executable. To find this information, it first searches through a text file stored on your machine called `desktop.sct`. If it cannot find an entry for that executable path of the component, it will ask you to find the file that needs to be launched and then will add an entry in `desktop.sct` for that component.

The `desktop.sct` file can be found under the user's home directory in the `.sctworkflow/config` directory. For example, `c:\Documents And Settings\aUser\.sctworkflow\config\desktop.sct`. A desktop application mapping applet is available in the Launching section of the User Information page to edit the contents of the `desktop.sct` file.

The `desktop.sct` file is updated by Banner Workflow, however, if you need to edit this file manually so it can be distributed it to several users, it is important to use the following format for elements in the file:

```
wordpad=C:\\WINNT\\NOTEPAD.EXE
```
Internationalization

Locale information is read from the http header. This allows users access to Banner Workflow based on the locale/language/encoding preferences stored in their browser.

Note

By default, dates are displayed in a locale neutral format of DD-MMM-YYYY, where DD represents the day of the month, MMM is the three-letter abbreviation for the month, and YYYY is the year. It is possible to specify locale-based date formats by editing the dateFormats.*properties in the ApplicationResources*.properties files that control locale-specific messages.

Look and Feel

Banner Workflow is deployed with two different 'styles'; the standard workflow style, and a luminis style. The stylesheets are generated during deployment (or regenerated using the wftool updateSystem command).

The template stylesheet is found in WORKFLOW_HOME/css/wfstyle_template.css. It is a parameterized file. The values contained in WORKFLOW_HOME/css/wf_stylesheet_token_dictionary.txt are substituted into it to create the standard Banner Workflow stylesheet. The values in WORKFLOW_HOME/css/luminis_stylesheet_token_dictionary.txt are substituted into it to create a stylesheet used for Banner Workflow when it is embedded in a Luminis tab or launched from Luminis.

You can edit the values in the two token dictionary files to change Banner Workflow colors, etc.

In particular, you will want to edit the @defaultLuminisColor values if deploying to Luminis Platform 4. (Banner Workflow ships with the values set to match the default colors of Platform III.) The luminis_stylesheet_token_dictionary.txt contains commented lines describing how to switch between default Platform III and IV colors.

Note

For changes made to the token dictionaries to take effect, you must rebuild the ear file (wftool updateSystem), and redeploy it.

Note

If you customize the token dictionaries, you should back them up somewhere outside the WORKFLOW_HOME, as they may be overwritten during patching/upgrades. After applying a patch or upgrade, you should manually compare the replaced dictionaries with your copies, and update the new versions as needed.
Banner Workflow provides a web service interface for integrating third party and custom applications into a workflow solution to retrieve work item context (parameter values), manipulate work item state, and post external events, as well as other operations.

A web service allows different machines to communicate over a network using existing web technologies. The Banner Workflow web service implementation adheres to standards defined in the *WS-I Basic Profile 1.1 Specification*. As such, Banner Workflow provides a Web Services Description Language (WSDL) file that defines the interface, binding, and location of the service.

The web service is developer friendly and will work equally well across different platforms. There are many tools that will take the wsdl file generated by Banner Workflow as input and create client stubs for exchanging messages. You can find examples of this in the `WORKFLOW_HOME\examples\ws` directory.

**WSDL and SOAP Messages**

Banner Workflow 8.0 supports Web Services version 1.1.

The WSDL file for the workflow instance can be viewed from your web browser using:

URI: `{protocol}://{hostname}:{port}/{webapp}/ws/services/WorkflowWS/v1_1?WSDL`

Where:

- `{protocol}` Either http or https.
- `{hostname}` The name of the machine or IP address of the workflow server.
- `{port}` The port number that the workflow application is running on.
- `{webapp}` The name of the deployed web application for the server.
The xml schema defining the messages passed to and from workflow is located in the messages.xsd file in the WORKFLOW_HOME/wsdl/v1_1/workflows directory, or you may access it from your web browser using:

URI: {protocol}://{hostname}:{port}/{webapp}/ws/wsdl/v1_1/messages.xsd

The Banner Workflow web service messages are defined in a document/literal style. The xml traffic will appear more as pure messages than in a remote procedural call format. This style of message is well supported across both .NET and Java.

**Endpoint**

URI: {protocol}://{hostname}:{port}/{webapp}/ws/services/WorkflowWS/v1_1

**Authentication and Authorization**

Each workflow web service request requires an authentication element that takes a principal (logon id) and a credential (password).

For backward compatibility, you can invoke the web services with the wfwebservices account. This will invoke the service as a superuser, allowing any functionality to be invoked. You would typically do this when the invoker is an automated process of some kind.

The other way to invoke the web services is with a regular workflow user account. In this case, the user's roles will be used to check whether or not the user is authorized to invoke the functionality, just as if they were working through the GUI.

For automated processes where you simply want to invoke workflow operations as a superuser, you can continue to use wfwebservices, although it is recommended that you create a real workflow user account instead, and assign the necessary permissions to the account. For cases in which you are doing work on behalf of a real workflow user, it is highly recommended that you invoke the web service with that user's credentials.

Failure to authenticate with the workflow web service will result in a NotAuthorizedFaultDetail fault being thrown.

**Operations**

The Banner Workflow web service defines a set of operations uniquely identified by the type of xml message (request) that is received from the client. Once the operation has completed processing the request, a corresponding xml message (response) or fault will be returned as declared in the wsdl.

The following operations (messages) are included with Banner Workflow as part of the 1.1 web services:
Data Passing:

- **getWorkItemContext**: Returns the parameter names and values that are stored in the work item.
- **setWorkItemContext**: Sets parameter names and values for a given work item.

State Manipulation:

- **completeWorkItem**: Marks the work item as complete. If successful, the workflow will transition to the next step in the workflow.
- **releaseWorkItem**: Releases the work item. The work item will be available again to any applicable work lists.

Event Creation

- **postExternalEvent**: Posts an external event. An external event is an event that occurs in a system outside of Banner Workflow, but maps to an event definition in the Banner Workflow system. Posting the event to the workflow system allows workflow to evaluate the event and start zero or more business processes based on the data contained in the event.

Account Provisioning and Manipulation:

- **createUser**: Create a Banner Workflow user account.
- **updateUserAttributes**: Update a Banner Workflow user account.
- **getUserAttributes**: Get attributes of a Banner Workflow user account.
- **updateUserAuthenticationRequest**: Update a user's authentication method and details (internal/external).
- **deleteUser**: Remove a user account.
addRoleAssignmentForUser

Add a role to a user.

findUsersWhoAreExternallyAuthenticated

Return all externally authenticated users.

findUserByExternalId

Find a user identified by a given ExternalId.

getExternalIDForUser

Return the externalId for a specified user.

Please consult the wsdl and messages.xsd file for the specific makeup of the xml messages.

Usage Notes

For messages that manipulate a specific work item, a work item primary key (workItemPK) needs to be provided. Typically this would be passed to the business component process via a command line style argument. @BUILTIN_WORKITEM is a macro that may be supplied as the value of a launch parameter for dynamically substituting the current work item primary key.

The data passing operations (getWorkItemContext and setWorkItemContext) are useful for all types of business components that you will create yourself. These operations will read input, do processing, and finally write the output.

For example, if you needed to create a simple adder component, you could define an automated business component that takes two required numeric parameters, for example first and second, and returns a guaranteed numeric parameter sum representing the summation of the two input parameters. You would pass along @BUILTIN_WORKITEM as a launch parameter to pass the work item primary key. In writing the actual program, you would read the work item primary key as one of the command line arguments. Using the key, your program would fetch first and second using getWorkItemContext. The program would then compute the result of adding first and second. Finally, the program would send back the result as the value for sum using the setWorkItemContext message.

When creating custom automated activities, it is usually sufficient to set the component type to automated non-workflow aware. The server will use the return code of the launched process to automatically mark the work item complete or create an alert for error handling.

The state manipulation operations (completeWorkItem and releaseWorkItem) are especially handy for interactive web forms as they may correspond to the submit and cancel buttons traditionally present in a web form. The state manipulation operations are also useful for creating automated workflow-aware components if you need that level of control.
The postExternalEvent operation is useful for placing event triggers directly into your application without relying on database table triggers which may not be applicable. Note that events attempt to start business processes which in turn may start workflows and events, business processes, and workflows need to be aligned in order to successfully instantiate a workflow remotely. See “External Events” on page 139 for details on the information an external event must contain and how that information is used to evaluate the event within workflow.

The workflow system is generally strongly typed. Parameter values that are passed using setWorkItemContext or postExternalEvent should use a specific xml type element (stringValue, booleanValue, dateValue, or numericValue) for properly representing a value. If the value is null, simply omit the value element since null is not considered to have a type. For postExternalEvent, you may optionally pass boolean, date, and numeric values as string values in string representation and allow the workflow system to determine the type.
Banner Workflow is compatible with high availability options offered by Oracle Application Server (Oracle AS) 10g. This chapter includes some guidelines for taking advantage of Oracle AS services and is meant as a supplement to the “Oracle Application Server High Availability Guide”. Not all options offered by Oracle AS are supported in this release of Banner Workflow.

**Benefits**

Oracle High Availability services seek primarily to remove single points of failure in the system. At the core of this are facilities in the application server to support the creation and management of Oracle Application Server clusters.

Clustering allows for fail over and promotes scalability. If a single node goes down, the system may continue to service requests if an additional node is up and available in the same cluster. A node could go down because of routine maintenance or a disabling event such as a loss of power. An application server node could also be overwhelmed with requests. In this case, scalability is improved by having additional nodes that can share the workload of servicing requests.

Clustering is often suggested as a solution for addressing performance. In some cases a cluster may improve scalability but reduce performance. Usually some time is taken for a load balancer to distribute the workload, and this will affect performance. But, since more nodes are active to service work requests, there is less likely a chance of the system stalling. This improvement in scalability is usually much more important in an enterprise environment, while the impact to performance is negligible.

However, if performance is a concern and the system load is relatively low, please be sure to inspect the system configuration parameters. Criteria such as the physical hardware, number of connections, number of threads, memory allocation, and other tunable parameters all have a strong bearing on the actual performance metrics. Increasing the performance often involves computing a benchmark, adjusting a parameter, and then recomputing the benchmark.
Deployed Banner Workflow Environment

At a high level, a workflow deployment consists of an enterprise application that runs in the Oracle Application Server and at least one workflow engine instance that runs as a process on the operating system.

The workflow engine manages the live processing of real workflow and work item instances by handling event processing and the automation of workflow state transitions.

The Oracle Application Server provides a set of server side components. Key components include:

- an http server for handling browser and http-based message requests
- a Servlet engine for rendering web pages
- a Java J2EE container for calling business logic and managing database requests.

Generally, the Oracle AS functionality plays a strong role in the user interface and services for managing the workflow administration (model time) data.

During Banner Workflow installation and deployment, an oc4j instance is created (usually with the name OC4J + install name). The oc4j instance contains the servlet and enterprise services used by the Banner Workflow application. For this segment of the system software, the oc4j instance performs the bulk of the processing and is the component that is principally clustered.

Both the Oracle AS and the Workflow Engine play key roles in defining the activity and responsiveness of the system. Each module has its own mechanism for defining a cluster and bringing multiple nodes into the system. If high availability is a goal, effort should be made to cluster both modules across multiple machines. When addressing scalability, please be conscious of the issue you are trying to address.

For example, a workflow engine cluster is more likely to assist with massive amounts of events being produced. Likewise, an Oracle AS cluster will better address the responsiveness to a large number of concurrent users being logged in.

Keep in mind that Banner Workflow services could be processing activities across different machines. If you have an externally run automated activity (e.g., a shell script), the external resource and the file path should be accessible from each node in the cluster.

Note

When running the configwizard, you should always specify the hostname and not refer to "localhost". This is especially true when specifying the location of a Workflow Engine instance. In particular, if you are using clustering, you should edit all engine instances in the configuration.xml file and ensure that they point to fully qualified hostnames and not "localhost".
It is critical to both Workflow Engine clustering and Oracle Application Server clustering that each node in the cluster have clocks that are synchronized to the exact same time. A good practice is to use specialized software that sets the clock according to an Internet Time Server.

The rest of this section focuses on Oracle AS clustering. Please see “Deploying and Managing Multiple Engine Instances” on page 43 for more information.

Basic Topologies

For planning out how you will cluster the Oracle Application Server piece, there are a couple of basic topologies involving the OracleHttp Server (OHS) component to keep in mind. Each Oracle AS 10g install contains an OHS component that may be enabled or disabled depending on the load balancing strategy. The load balancer decides how http traffic is distributed among nodes in the cluster. In the first topology, a single OHS instance is used as the load balancer. In the second, a dedicated load balancer (usually hardware) is used to distribute the load.

Single Oracle Http Server

In this topology, a single Oracle AS 10g instance is added to the farm but not the cluster. The administrator registers each node in the cluster and sets a load balancing policy in the mod_oc4j.conf file. The mod_oc4j.conf is the configuration file for the mod_oc4j adapter that allows the Oracle Http Server to communicate to OC4J instances. Starting the OHS components in the clustered instances is optional since traffic will generally go all through the single OHS outside the farm to the mounted OC4J nodes. This deployment is relatively inexpensive and appropriate for solutions that are risk-tolerant of using a single OHS process as single point of failure.

Hardware Load Balancer, Multiple Oracle Http Servers

In this topology, a hardware load balancer is added to route requests to OHS instances in the cluster. This solution offers increased high availability and scalability for heavily accessed http servers that may be taken down. In this case, the hardware load balancer becomes the single point of failure, but is generally considered more robust than software load balancing.

In either scenario, all the http requests in a single http session should be routed to the same OC4J instance. If the OC4J instance is down for maintenance or because of an error condition, a user logon session will need to be re-established (seamless fail over of http sessions via web replication is not supported in Banner Workflow).
Most load balancers have some provision for dictating how work is distributed. A round robin distribution is the default. As is the case with OHS + mod_oc4j, you can assign weights or server preference if particular nodes have greater resources than another.

**Managed Oracle Application Clusters**

Clusters of Oracle Application Server instances are typically defined and managed from services provided by the Oracle Application Server instances themselves. Oracle automatically deploys the application to all other nodes that are part of the cluster. You can use Enterprise Manager to act on the cluster as a single unit or an individual node, for example, to bring down a service for maintenance.

Regardless of which release of Oracle AS is being used, you will need to select a mechanism for how individual Oracle AS nodes identify themselves to the cluster. In most cases, you will need to know this information before installing Oracle AS.

*Note*

Each node in a managed cluster is required to be on the same operating system platform, though the farm can span platforms.

A manually configured Oracle Application Server Cluster is simply a collection of application server instances manually loaded and configured similarly. Like a managed cluster, you will need to register the instances so that they may be shared by a load balancer. Oracle tools such as the Enterprise Manager and dcmctl utility will not recognize such a collection as a cluster, therefore, activities such as configuration changes and server restarts will have to be applied specifically on each instance.

**Oracle Application Server Releases**

Oracle made a significant shift in how clusters are formed and administered between releases 10g R2 and 10g R3. The new R3 release is generally simpler and easier to deploy. Less useful concepts such as the application server farm which is required in R2 are no longer required with R3.

Please consult the “Oracle Application Server High Availability Guide” for specific details relevant to your release of Oracle AS.

The rest of this chapter will highlight installation and configuration options that you will need to consider under both releases.
Oracle Application Server R3 Clusters

Oracle AS 10g 10.1.3.4 Installation

It is recommended that you select Advanced Install when proceeding so that you have control of which application components and settings are applied. In most cases, you will want to select “J2EE Server and Web Server” as the “Installation Type” but this will vary depending on your desired topology.

On the “Administration Settings” view, Select “Configure this as an Administration OC4J Instance” to enable and run the Enterprise Manager for the cluster at this location. Oracle recommends only configuring one instance in the cluster topology with this flag enabled.

Next, you will be presented with the “Cluster Topology Configuration” view. This view is for entering a multicast address for identifying Oracle AS nodes to the cluster. Enter this value only if you have decided to use a multicast address across each node and have decided on the ip address and port to use. The value can be set after the installation.

Cluster Creation

There are two main ways to create a cluster in Oracle AS 10g R3:

- Dynamic Discovery Method (multicast)
- Discovery Server Method.

In the Dynamic Discovery Method, you define a common multicast address and port for each Oracle AS instance participating in the cluster. The benefit of this approach is that you do not have to register the name of each Oracle AS instance in the cluster.

You can set up multicast for each instance in the cluster from the command line as follows:

```
ORACLE_HOME/opmn/bin/opmnctl config topology update
discover="*225.0.0.20:8001"
```

Ensure that you reload the opmn after making this change by executing the following command:

```
ORACLE_HOME/opmn/bin/opmnctl reload
```

In the Discovery Server Method, you define the cluster by specifying a reference to a discovery server as a list in the opmn.xml file of each instance. The discovery server can be any Oracle AS instance in the cluster that you designate, and you may designate more than one node to be a discovery server.
You will need the host name (or ip address) of the discovery server(s) and the port that the discovery server(s) is accepting requests on. You may get the port value by opening the opmn.xml of the discovery server, and viewing the request attribute of the <port> element under the <notification-server> element.

In the opmn.xml file for each instance in the cluster, specify the nodes that are running the discovery server(s) using the <topology> and <discover> elements.

<notification-server>
... 
<topology>
<discover list="node1.sungardhe.edu:6200, node2.sungardhe.edu:6200"/>
</topology>
</notification-server>

Ensure that you reload the opmn after making this change by executing the following command:

```
ORACLE_HOME/opmn/bin/opmnctl reload
```

**Groups**

You can create and manage groups from the Enterprise Manager. Oracle AS 10g R3 uses the concept of a group, a named collection of OC4J instances, for performing operations such as start, stop, and restart and deployment operations. Groups may span every instance of the cluster or only a subset of instances.

**Deploying to the Cluster**

Because the Banner Workflow deployment scripts modify contents of the opmn.xml file directly, deployments scheduled after the cluster has been established need to be conducted using the Oracle Enterprise Manager interface.

When creating a new OC4J instance, you always specify the group that it belongs to. By convention, the group name should be the same suffix as the application you are deploying. The OC4J instance will be created on all instances in the cluster which are members of the group. You may also specify a new group name at the time of OC4J creation and adjust the instances tied to the group afterwards.

Use the Enterprise Manager to deploy the workflow ear to the specified group. This can also be done using the admin_client.jar utility from any member of the cluster.

See “Re-deploying after configuration changes” on page 35 for more information on manually deploying the workflow ear file to an oc4j instance.
If the opmn.xml file needs to be modified for the case enabling SSL, drill into the Oracle AS 10g Application Server page for that particular instance and click Process Management. This link will take you to a page where you can modify the contents of the opmn.xml file and keep the system in a consistent state.

**Configuring Oracle Http Server to Load Balance.**

There are no special modifications needed to the Oracle Http Server to allow it to distribute requests across members of the cluster. Consult the manual, Oracle Application Server High Availability Guide 10g Release 3, for information on setting the load balancing algorithm if you wish to change the default behavior.
An external event is an instance of an event that originates outside of Banner Workflow and can be mapped to a Workflow Event. An external event can be posted to a Banner Workflow system, which causes Banner Workflow to persistently record the details of the event and queue it for evaluation. Zero or more workflows will be started as a result of evaluating the external event.

Consumption of external events by a workflow system is divided into two distinct phases, with certain responsibilities involved in each phase. In the first phase, an event provider posts the external event to the workflow system. During the post, Banner Workflow will persistently store a copy of the external event, and schedule it for evaluation. Once the post succeeds, the event provider will consider the external event to have been handled, and is no longer responsible for the external event in any way.

**Note**

The event provider has no knowledge of whether the external event evaluates successfully or not. The only responsibility of the event provider is to notify Banner Workflow of the existence of the external event and the data contained within it.

A successful post means that Banner Workflow is guaranteed to evaluate the event at some future point, but makes no guarantee as to when that will happen or the exact order in which it will occur, since external events are not guaranteed to be processed exactly in the order in which they originate in the external system. In practice, in a well-tuned system event evaluation will occur within seconds of a post.

External events may be posted to Banner Workflow by SunGard Higher Education delivered event provider classes, such as the `BannerTableExternalEventProvider`, or they may be posted by custom code via Web Service interfaces to the `ExternalEventService`. The `ExternalEventService` is responsible for persistently storing the details of the event, detecting duplicate events, and scheduling posted events for evaluation by an engine node.

To simplify development of custom code that posts external events to Banner Workflow, Banner Workflow itself takes responsibility for detecting and rejecting duplicate events. This allows multiple event providers to be run for the same external event source without the providers themselves having to provide a mechanism to screen out duplicate events. For example, if the source of events is a JMS topic, you can have two programs subscribe to the same topic, both feeding the same events to Banner Workflow in order to provide transparent failover. Banner Workflow will automatically reject the duplicates from the second provider.
The second phase is the evaluation phase. During the evaluation phase, a workflow engine node evaluates the posted event, and starts zero or more workflows on its behalf, or, if errors are encountered, raises the appropriate alerts to the administrator of the Banner Workflow system. Any errors encountered during event evaluation must be dealt with within the Banner Workflow system. The event cannot be reposted from the external system, since Banner Workflow will recognize it as a duplicate and reject it. Banner Workflow provides various pages to monitor the state of external events that have been posted, as well as to replay ones that have encountered errors. Banner Workflow also maintains a log of event evaluation details for successful evaluations that can be accessed via the user interface.

The evaluation is either successful, and all workflows that are supposed to start do so, or all work is rolled back, and an alert condition is raised for the event. At this point, a workflow administrator or analyst can correct the problem, and request the event to be evaluated again, without any fear of starting duplicate workflows.

**External Event format**

In order to successfully post to the Banner Workflow system, an external event must contain several pieces of information.

For Banner Workflow to detect duplicate events, each external event must have a unique ID. This unique ID is actually a 2-part key consisting of a *source ID* and an *external ID*, both of which are strings of up to 255 characters. The source ID is a name used to uniquely identify an external system, for example, *BannerProduction*. The external ID is then an ID for the external event instance in that external system. The external ID should be unique for each event instance produced by the system identified by source ID. This allows Banner Workflow to uniquely identify every event instance posted from any external system.

**Note**

If you are posting events from an external system that does not or cannot provide unique IDs for events, then Banner Workflow cannot guard against evaluating the same event more than once, and the custom code posting the events must ensure that each event instance is posted exactly once.

In addition to the source ID and external ID needed to identify the event, the external event must also carry the name and product type of the Workflow Event it maps to. This is the event as defined in the Banner Workflow system, and that is mapped to processes and workflow definitions with guard conditions, and determines what workflows, if any, should start as a result of the event.

The external event may optionally specify a name for any workflow instances that are started as a result of the event.
The external event can contain any number of *name/value* pairs containing information specific to the event that will be used in guard condition evaluation and/or passed on to the workflow as initial context data.

The Web Services interfaces require that typed values be passed for the parameters. However, to maintain backward compatibility, and to support systems that cannot provide type information with the parameters, all of the parameters can still be passed as string representations (using *Text*) type for the value. When the Banner Workflow system attempts to evaluate the event, it will automatically attempt to convert incoming *Text* values to the type defined on the parameter in the event.
Authentication

Banner Workflow provides support for integrating with some external systems for authenticating users. When configured to use external authentication, Banner Workflow can delegate the process of authenticating a user's identity and credentials to an external system.

By default, within Banner Workflow a user account is identified by a unique logon ID and protected by a securely stored password. The account contains all of the user's roles and preferences within the Banner Workflow application.

When external authentication is configured, any Banner Workflow account may be designated as externally authenticated. The enabled, effective dates, roles, and preferences for the Banner Workflow account will still be stored in Banner Workflow, but the user's password will not be. Instead, when a user attempts to logon to the system, the authentication process will be delegated to the external authentication service (EAS). If a user successfully authenticates with the EAS, then they will be logged into Banner Workflow with their associated roles.

Since accounts can be individually marked as being externally authenticated, you can still create local accounts that authenticate against Banner Workflow directly. For example, it may be useful to have local admin accounts when setting up and configuring Banner Workflow, while the majority of the user accounts are set for external authentication.

**Note**

An externally authenticated user will not be able to log into Banner Workflow if the EAS configuration is incorrect, or if the EAS is not running.

Since externally authenticated users will logon to Banner Workflow with their external username and password, Banner Workflow must provide a mechanism to link the external account with the Banner Workflow account. This is accomplished by setting the external ID on the Banner Workflow account to a value that can be used to uniquely link the account to a user in an external system. The actual value needed in the external ID will vary depending on the type of external authentication being used.

When a user attempts to logon to Banner Workflow, the system will evaluate the logon in the following order:

1. does the username represent a built-in account (e.g. wfroot)? If so, authenticate as a built-in account

2. does the username identify an internal user account? If so, authenticate as an internal user
3. otherwise, if external authentication is enabled, delegate to the EAS

This has several important consequences. If a user account is externally authenticated, then the logon ID specified in the Banner Workflow account is meaningless for logon purposes; an externally authenticated user must logon to Banner Workflow using their username/password from the external system.

Furthermore, since Banner Workflow gives precedence to internal accounts, an internal account having a logon ID equal to the username of an external account will always block the external user from logging in.

For example, consider two workflow accounts. The first is internal and has a logonID=jsmith.

If there is a separate external user with (external) username jsmith, the external user can never logon to Banner Workflow, since Banner Workflow will always treat a username of jsmith as identifying the internal account.

For this reason, SunGard HE recommends that limit your use of internal accounts as much as possible if you are taking advantage of external authentication. If possible, use a unique prefix not present in the username of any external accounts (e.g. "workflow-") on all internal account logonIDs to minimize the risk of overlap.

Banner Workflow currently ships with support for two different external authentication modes:

- Banner/Oracle authentication
- LDAP version 3

Note
As of Banner Workflow 8.0, the two LDAP modes from earlier releases have been combined into a single, more flexible external authentication adaptor.

Note
You can only configure Banner Workflow to use a single mode of external authentication.

To enable external authentication, you need to edit the `<ExternalAuthenticator>` element under `<SecurityIntegration>` in the `configuration.xml` file. For more information, see “configuration.xml” on page 100.

After you configure the `<ExternalAuthenticator>` element, you can test it on the server by using the `checkexternalauth` script. Give this script a valid username and password in the external system, and it will attempt to authenticate it the same way Banner Workflow would if that account was mapped to a Banner Workflow user. For example:

```
bin/checkexternalauth jsmith password
```
External Authentication

Before selecting and configuring an external authentication mechanism, it is helpful to understand how Banner Workflow proceeds when authenticating an external user.

At the start of the sequence, workflow has been presented with a username/password pair (for the external system), and must perform 2 tasks:

1. verify that the password is correct for the username (the actual authentication part)
2. determine which Banner Workflow user is identified by the external username

Since the username in the external system may be mutable, it is desirable to be able to 'link' a Banner Workflow User to an external user on some immutable value. On the Banner Workflow user, this value is stored in the externalID field (sometimes this is also referred to as the 'linking attribute').

Note
In the Banner Workflow UI, the externalID field is the box under the External Authentication option on the User Management screen.

The actual value used to link the accounts varies depending on both the external authentication mechanism, and how accounts are stored in the external system. Ideally, the externalID should link the accounts on an immutable value, so that there are no account synchronization issues.

Banner Authentication

This method of external authentication uses the username and password presented to Banner Workflow to open a connection to the database specified in the <BannerDataSource> element under <Deployment>. If the external ID and password can successfully open a connection, the user is considered to be valid, otherwise, the user is rejected. The username is used as the externalID to identify the Banner Workflow account. (That is, if you use Banner Authentication and successfully authenticate as 'jsmith', Banner Workflow will log you in as the Banner Workflow user having externalID=jsmith. You should use this method of external authentication when your Banner Workflow users are also Banner users, and you are not integrating Banner Workflow with Luminis, and want to avoid having to keep Banner Workflow and Banner passwords in sync.

To enable the Banner authenticator, edit the <ExternalAuthenticator> element under <SecurityIntegration>. Set the <ClassName> element to com.sungardhe.workflow.security.BannerAuthenticator. You do not need to set any properties for this authenticator.
For example:

```xml
<Authentication mode="WorkflowExternal">
  <WorkflowExternal>
    <ExternalAuthenticator>
      <ClassName>com.sungardhe.workflow.security.BannerAuthenticator</ClassName>
      <Properties/>
    </ExternalAuthenticator>
  </WorkflowExternal>
</Authentication>
```

**LDAP Support**

For Banner Workflow to be able to authenticate against LDAP, it must be able to translate a username into the Distinguished Name (DN) of a user in the Directory Service, authenticate using the provided password, then use an attribute of the LDAP account to identify the corresponding Banner Workflow account for the user.

**Search**

When using LDAP Authentication, Banner Workflow will use the username presented during logon to execute an LDAP search to find the user’s account in the LDAP Server. The user accounts can be in the same container, or stored in separate containers.

For example, suppose we have users stored in LDAP in two different containers, one for finance users and one for human resources users. Also assume a Luminis Platform 4 layout in which the users are assigned a unique, immutable ID (uid) that is part of their distinguished name, and that their username (that they logon to systems with) is stored in a logonID attribute.

```
o=myschool
  |   o=people
  |     o=finance
  |       uid=12345
  |       logonID=johnsmith
  |     o=hr
  |       uid=56789
  |       logonID=joansmith
```

The following must be true:

- The subtree containing all external users that need to logon to Banner Workflow must be anonymously searchable, or you must create a dedicated (LDAP) account that Banner Workflow can use to search the subtree.
• It must be possible to locate each user from a query into which the username has been substituted.

In the above example, the subtree searched is `o=people, o=myschool`. Assume that the tree is not anonymously searchable, but that an account `uid=wfsearcher, o=myschool` with a password of `password` has been created and given permission to search the `o=people, o=myschool` subtree.

Since we are referring to a Luminis Platform 4 environment, the usernames are stored in the logonID attribute, and the uid attribute contains the immutable identifier for the user. In this case, the uid attribute is the best choice for the 'link attribute' to link the LDAP account to a Banner Workflow user, as it won’t ever change.

**Note**

In Luminis Platform III, the usernames are stored in the uid attribute and cannot be changed.

If johnsmith and joansmith are both workflow users, then we would create their accounts in workflow, and give them externalIDs of 12345 and 56789, respectively.

To configure Banner Workflow to use external authentication in this case, the `<ExternalAuthenticator>` element is configured as follows:

```xml
<Authentication mode="WorkflowExternal">
  <WorkflowExternal>
    <ExternalAuthenticator>
      <ClassName>com.sungardhe.workflow.security.LDAPSearchAuthenticator</ClassName>
      <Properties>
        <Property name="java.naming.factory.initial" value="com.sun.jndi.ldap.LdapCtxFactory"/>
        <Property name="java.naming.provider.url" value="ldap://myschool.myhost.edu:389"/>
        <Property name="directory.user" value="uid=wfsearcher,o=myschool"/>
        <Property name="directory.user.password" value="password"/>
        <Property name="search.directory" value="o=people,o=myschool"/>
        <Property name="search.filter" value="(logonID={0})"/>
        <Property name="link.attribute" value="uid"/>
      </Properties>
    </ExternalAuthenticator>
  </WorkflowExternal>
</Authentication>
```

In the above example:
<ClassName>

Tells the system what external authenticator to use, in this case, the authenticator implemented by the com.sungardhe.workflow.security.LDAPSearchAuthenticator class.

<Properties>

Defines a collection of name-value pairs used to configure the LDAPMappingAuthenticator.

java.naming.factory.initial

Defines the context factory to use, and should not be changed except on the advice of SunGard Higher Education.

java.naming.provider.url

Defines the url to your LDAP server. In the example above, the authenticator will connect to server myschool.myhost.edu on port 389.

directory.user

The distinguished name of an LDAP user that has permissions to search the search directory.

directory.user.password

The password of the LDAP user that has permissions to search the search.dn tree.

search.directory

The distinguished name of the entry at the top of the subtree that contains all the users that need to authenticate within Banner Workflow.

search.filter

An LDAP search filter, expressed as described in javax.naming.directory.DirContext, that should be executed to find the LDAP account. The username presented to workflow for authentication will be substituted anywhere '{0}' occurs.

link.attribute

The name of the LDAP attribute on the found account that links the LDAP user to a Banner Workflow user. (Matches the externalID field of a Banner Workflow user.)

Now when johnsmith attempts to logon to Banner Workflow using johnsmith as his username, Banner Workflow will logon to the Directory Server as uid=wfsearcher, o=myschool and search the o=people, o=myschool tree using the search filter ‘(logonID=johnsmith)’. It will find the unique entry uid=12345, o=finance, o=people, o=myschool, which represents Jonh Smith's LDAP account, and then will attempt to bind to LDAP as uid=12345,o=finance,o=people,o=myschool using the password he presented at logon. Assuming he provided the correct password, the bind will be successful. Banner Workflow will next extract the value of the link.attribute (in this case, uid) from the LDAP entry, then search for a Banner Workflow account having the uid
value (12345) as the externalID. At this point, Banner Workflow has successfully authenticated the user, and identified his account within Banner Workflow.

**Note**
If there were also a uid=98765, o=hr, o=people, o=myschool, with logonID=johnsmith, the logon would fail, as the search would find two separate LDAP accounts identified by the johnsmith username.

**Note**
The LDAPSearchAuthenticator uses JNDI to access the Directory Server, and will pass any properties defined in the `<Properties>` element into the JNDI InitialContext object used to make the connection. Consequently, any other JNDI properties you may wish to pass to the JNDI context may be set here.

### Security

If the communication between the Banner Workflow server and the Directory Server is vulnerable, SunGard Higher Education recommends that SSL be used to secure the connection. SSL can be enabled for the LDAPSearchAuthenticator by adding the property `java.naming.security.protocol=ssl` to the configuration. For example:

```
<Property name="java.naming.security.protocol" value="ssl"/>
```

**Note**
If you enable SSL, you will need to import the LDAP server's certificate into the trusted store for the Banner Workflow appserver. Follow Oracle's documentation to import the LDAP server's certificate as a trusted certificate.

### SSO Authentication

Banner Workflow supports the following two authentication modes for SSO integration:

- **CAS** - Workflow will delegate authentication to the CAS (Central Authentication System) 2.0 server that has been extended to include the UDC ID to support the Banner Validator service specified in the configuration. If the user is not logged in, Workflow will redirect the user to the CAS login page. Please refer to “Configure Banner Workflow for CAS” on page 150 for details on CAS configuration.

- **IDM gateway** - The IDM gateway is responsible for securing access to Workflow. Workflow will use the identity cookie or header passed by the IDM gateway to identify which Workflow user is requesting access. Please refer to “Configure Banner Workflow for IDM Gateway” on page 151 for details on IDM Gateway configuration.
UDC ID Contract

IDM Gateway - Workflow requires that the UDC ID of the Workflow user be inserted into a HTTP header or cookie. When IDM Gateway mode is enabled, it is assumed that the IDM Gateway is acting as an access control gateway to control access to the Workflow server. Access to Workflow can be granted for any active user if they have an external ID, since Workflow directly assumes the identity of the user from the external ID present in the HTTP header or cookie.

CAS - Workflow requires that the external ID of the Banner Workflow user be set to the UDC ID of the Banner Workflow user.

Configure Banner Workflow for CAS

To integrate Banner Workflow with CAS, set the following configuration properties:

- **ServiceURL** – The Workflow URL used to service requests from the CAS server. Should be `<protocol>://<workflow host>:<workflow port>/<workflow root>/j_spring_cas_security_check`. For example:
  
  http://school.edu:7777/workflow/j_spring_cas_security_check

- **LoginUrl** - The URL to login to the CAS system

- **AuthenticationProviderKey** - Key required by Workflow's CAS authentication provider so that it can identify tokens that it previously authenticated

- **ProxyTicketValidatorUrl** - The URL of the CAS system. For example:
  
  http://school.edu:7777/cas

Configure IDM Gateway

To configure the IDM Gateway for use with Banner Workflow ensure that the IDM Gateway is be configured to protect everything under workflow root except the following URL patterns:

- `/bannerintegration`
- `/channelservlet`
- `/j_spring_security_check`
- `/applets`
- /remoting
- /isSessionAuthenticated
- /ws/services

Note
In addition, please ensure that the Identity Management software is not injecting no-cache into the header. Having no-cache in the header will prevent certain workflow components, such as the Workflow Modeler, from launching.

Configure Banner Workflow for IDM Gateway

To integrate Banner Workflow with the IDM Gateway, set the following configuration properties:

- Source - “Header” or “Cookie”
- HttpVariableName - The name of the HTTP header or cookie (depending on the Source attribute) variable that the IDM Gateway will populate with the UDC ID of the current user

Configure the <LogoffUrl> element

When Workflow is running in CAS or IDM Gateway mode, clicking the Logoff link will not log the person out of Workflow because they will still be logged into the CAS or IDM Gateway.

An optional <LogoffUrl> element can be set to provide a provide a link to an external URL to log the user out of the external CAS or IDM Gateway system.

Note
The <LogoffUrl> must be the last child element of <SecurityIntegration>.

Configure Luminis for Banner Workflow

If SafeUTFURL is enabled (used to prevent XSS attacks), you must either add Banner Workflow to the allowed list or disable SafeUTFURL.

Example: if Banner Workflow is deployed to a host named sample.sungardhe.edu using http, then it may be necessary to manually add the sample.sungardhe.edu to the allowed list in both normal and in encoded form:

```bash
configman -s com.pipeline.web.SafeUTFURL.url.0=http://sample.sungardhe.edu
configman -s com.pipeline.web.SafeUTFURL.url.1=http%3A%2F%2Fsamplesungardhe.edu
```
Log files are one way for a service to communicate with an administrator. They provide a persistent means for capturing messages and the timing of events in the system. Log files also expedite the process of tracking errors in your workflows. The Banner Workflow Server makes extensive use of logging. As such, it is a good idea to periodically inspect the logs to verify that the system is running smoothly.

### Banner Workflow Logs

When using the wftool script all logs generated will be located within `WORKFLOW_HOME/logs/wftool.log`. This log will provide an audit trail of the commands executed from the `wftool` script.

### Engine Logs

Each engine node contains its own log file. By default, one engine node is created in `WORKFLOW_HOME/engine`.

#### `WORKFLOW_HOME/engine/engine.log`

Contains log messages that will be related to the processing of work from one activity to another. This log will grow to 20 MBs and create up to 5 roll over files to prevent the disk from filling up. Default logging is ERROR except for the main engine services loop. This loop is set to INFO to provide more detailed information during startup and shutdown. You should examine this log file when the following situations occur:

- Activities are not progressing from one step to another - This typically happens when the Workflow engine is not started. If work is not progressing, check to see that your engine is running and secondly investigate this log file.

- Activity Notification Emails - If notifications are not being sent, the first place to check for error messages would be the engine log files. The email notifications are generated from the engine tier and if they fail are logged with the appropriate message.

#### `WORKFLOW_HOME/engine/install.log`

Provides the install log that was generated when installing an engine node. Typically this only has value when debugging an installation of an engine node.
Oracle Application Server Logs

All workflow specific log messages are directed to the Oracle Application Server logs. In most cases the Oracle logs are used to debug situations where deployments to the Oracle Application Server have failed. If, while running an automated deployment, you receive an *ADMN-xxxxxxx* error message, the Oracle logs may be able to provide more detailed information. Oracle has numerous log files but only a few are particularly useful when debugging deployments. The following is a list of Oracle logs and their significance:

**ORACLE_HOME/dcm/logs/dcmctl_logs/log.xml**

Every command that is run through the DCMCTL utility is logged in the log.xml file. All errors that occur will be listed here.

**ORACLE_HOME/opmn/logs/<OC4J Instance Name>.default_island.1**

This log contains the output generated from an individual OC4J instance. This log will typically be the most beneficial in deducing what caused a workflow to fail. By default, all workflow log messages from the web tiers are logged here.
The following is a description of all scripts the workflow installer creates under the /bin directory.

**archive**

Moves data on completed/stopped workflows to the archive tables. It takes the following arguments, in order:

- **username** - A valid workflow account or the wfroot account.
- **password** - The password for the account.
- **verbose** - Accepts 'true' or 'false'. If true, will write verbose output during processing.
- **displayContext** - Accepts 'true' or 'false'. If true, workflow context parameter values will also be archived.
- **days expired** - Archives workflows that have ended more than this number of days in the past.

You can also pass an optional -batchSize <size> option to control the sizes of the batches in which workflows will be archived. By default, the batch size is 50.

The following example will archive all workflows that stopped running 20 or more days ago, providing verbose output, storing parameter values, and deleting the workflows once they have been archived:

    archive admin password true true 20
**checkexternalauth**

A shortcut script used to check the setup of External Authentication. If you have configured your system to use external authentication, then you can run this script and provide it with an external ID and password. The script will use the external authentication mechanism to authenticate the given user. If it is successful, you know that the configuration is correct. It takes the following arguments in order:

- **username** - A valid external ID recognized by the external system responsible for authentication.
- **password** - The password for the username.

If you have configured workflow to use External Authentication, and a valid externalID/password is admin/password, then you can test it using the following command:

`checkexternalauth admin password`

If the settings are correct, then you will get a message indicating that the user was successfully authenticated, otherwise you will get an error message or a warning that the user could not be authenticated and should check your configuration settings.

**configwizard**

Runs the configuration wizard that prompts you to provide values for common configuration settings and automatically applies them to the configuration.xml file.

**export**

Exports administrative objects (users, roles, process definitions, etc) to an xml or zip file. It can be used to take a snapshot of the non-runtime elements of your system, or to move objects from one workflow system to another. It takes the following arguments in order:

- **username** - A valid workflow username or wfroot.
- **password** - The password for the user.
- **filename** - The name of the file to export to.

The export script also accepts the following options, which may be specified in any order:

- **-verbose** - Print verbose error messages that are useful for debugging.
- **-xml** - Exports only the content that can be represented in an xml format.
• `-zip` - Exports to a .zip file that includes binary content, such as global attachments and icon images. This is the default behavior unless the `-xml` option is used. The `-zip` option is maintained for backward compatibility with any scripts that may use it.

• `-primaryKey` - Export Primary Keys for each object. This should only be used when explicitly directed by migration instructions.

• `-maxThreads` - Sets the maximum number of concurrent threads to use when exporting objects that can be processed in parallel. The default is 2. It should generally be set equal to the number of processors on the server that is running workflow.

### `extractwd`

A script that can be run against an export xml or zip file to extract a Workflow Definition, along with supporting objects such as roles, components, and technology types. You can use this script to extract single workflow definitions from a test or development system to move them to a production system.

It requires the following options:

• `-source <source file>` - The xml or zip file containing the original export you want to extract the definition from.

• `-target <target file>` - The xml or zip file you want the extraction placed into.

• `-processdef <organization> <name> <version>` - Where organization is the qualified name for the organization of the definition. You may specify more than one `{<organization> <name> <version>}` sequence.

```
extractwd -source export.zip -target approvals.zip Approvals 0
```

The above example will extract the Approvals workflow from the export.zip file and place it within the approvals.zip file. This file can later be used to import into another Banner Workflow system.

For interactive workflow definition extraction the `-shell` option can be passed. This will prompt the user for the target, source, workflow definition name and version. For example:

```
extractwd -shell
```

**Note**

Since this script only processes an xml or zip export file, it is not necessary to have the workflow server running to use it.
import

A script used to import objects into the system. It takes the following arguments:

- **username** - A valid workflow username or wfroot.
- **password** - The password for the user.
- **filename** - The name of the file to import.

The import script also accepts the following options, which may be specified in any order:

- `-verbose` - Print verbose error messages that are useful for debugging.
- `-primaryKey` - Import Primary Keys for each object. This should only be used during migration when directed to by migration instructions.
- `-maxThreads` - Sets the maximum number of concurrent threads to use when importing objects that can be processed in parallel. The default is 2. It should generally be set equal to the number of processors on the server that is running workflow.

The import script will only create new objects, it will not modify or replace existing ones. If errors are encountered, import will continue to run, and attempt to successfully import as many objects as possible.

purgewf

A script used to delete both running and completed workflows from the system. By default, this script purges only completed/stopped workflows. It takes the following arguments:

- **username** - A valid workflow username or wfroot.
- **password** - Password for the username.

The purgewf script also accepts the following options, which may be specified in any order:

- `-workflowOrg` - Purges workflow instances started at the given qualified organization.
- `-processName` - Purges workflow instances with the given business process name.
- `-definitionName` - Purges workflows started for the workflow definition with this name. This option accepts wildcards ('%').
- `-definitionVersion` - Purges workflows started for the workflow definition with this version number. This option should be used with `-definitionName`.
• `-startDateFrom` - Purges only workflows having a start date greater than or equal to this value.

• `-startDateTo` - Purges only workflows started on or before the given date.

• `-endDateFrom` - Purges only workflows that have completed/stopped after this date.

• `-endDateTo` - Purges only workflows that have completed/stopped before this date.

• `-instanceName` - Purges only workflows having the specified instance names. This option accepts wildcards ('%').

• `-state` - Purges only workflows in the specified state. The following states can be specified:
  • started
  • stopped
  • started.running
  • stopped.completed
  • stopped.aborted

• `-noprompt` - Starts purging workflows immediately without prompting for confirmation.

**Note**
The values of the date format must be in one of the following accepted date formats (expressed using Java (TM) conventions):

- `dd-MMM-yyyy` - (20-JAN-2006, note that the time defaults to 12:00:00 AM)
- `dd-MMM-yyyy hh:mm:ss a` - (20-JAN-2006 01:00:00 PM)
- `dd-MMM-yyyy'T'HH:mm:ss` - (20-JAN-2006T13:00:00)

**Note**
All of the above options are considered to be joined by 'and' when more than one is used. For example, `purgewf wfroot password -workflowName mydef -workflowVersion 1 -startDateFrom "10-Mar-2004 10:00:00 AM" -state started.running` will delete all workflows started for the workflow definition 'mydef' version 1 that were started on or after 10 AM, March 10, 2004 that are still running.
Tool used to perform workflow installation and deployment tasks. The wftool accepts the following commands:

- **all** - This performs a complete installation sequence from scratch. You will be prompted to have all tables dropped and recreated. All workflow artifacts, including the ear file, will be rebuilt.

- **bannerdb** - Runs the sql scripts to redeploy Banner integration tables/components.

- **builddb** - Runs the sql scripts to redeploy all Banner Workflow and Banner integration tables/stored procedures. It should only be used during an initial install, or if you wish to delete all data from a test system.

  **Warning**
  
  This command will result in the deletion of all data in the system.

- **car** - Builds/rebuilds the Banner Workflow car file using the latest values in `configuration.xml`.

- **deleteAllWorkflows** - Deletes all workflows in the system. This should only be used in test systems.

- **deploy** - Performs an automated deploy of the ear file to the application server.

  **Note**
  
  This command will not build/rebuild the ear file. If you have made configuration changes, you should use `updateSystem` to rebuild the ear file before deploying.

- **ear** - Builds/rebuilds the workflow ear using the latest configuration values.

- **engine** - Builds/rebuilds the default engine node and engine installer.

- **scripts** - Creates the scripts in the bin directory.

- **updateSystem** - Rebuilds all workflow artifacts (the ear file, engine installer, car file, etc) using the latest configuration values and uploads the configuration to the database.

- **uploadconfig** - Encrypts and uploads the configuration.xml to the database.

- **workflowdb** - Runs the sql scripts to recreate all the workflow tables.

  **Warning**
  
  Running this command will delete all data in the system. It should only be run during an initial (fresh) install, or if you wish to delete all data in the system.

- **ziplogs** - Gathers various log files into a single zip file.
wftool supports the following options which may be specified after any of the commands:

- **-noprompt** - This will cause the command to automatically answer 'yes' to any yes/no question. Take care when using this with database related commands. For example, running 'wftool workflowdb -noprompt' will cause all data in the system to be deleted without any further input from the user.

- **-oracleHome** - For commands that require the path to the Oracle Application Server home (such as deploy), you can specify the oracle home with this option if you do not have an ORACLE_HOME environment variable set.

- **-help** - Displays help on all wftool commands and options.

## Engine bin Directory Scripts

The following scripts are created in the bin directory of each directory that an engine instance is installed into.

### startengine

This script starts the instance of the Workflow Engine installed in the directory.

### engineconsole

This script executes the engine console. The Console can display the status of all the installed engine instances, and can be used to pause, resume, or stop any of the instances. The console can be used to execute a single command, or it can be run in an interactive mode in which multiple commands can be executed.

To run the console in an interactive shell, execute 'engineconsole shell'. You can quit the shell by entering 'quit'. You can type 'help' to display help on the commands.

The console supports the following commands, in both command line or shell mode.

### stop

The following script will instruct one or more engine instances to stop running:

```
stop -password <password> [-all] [-host host] [-port port] [-force seconds]
```

Options:

- **-password**: Required option, the administrative password for the engine instances.
- **-all**: Instructs the console to send a shutdown signal to all known engines (all engines listed in the <EngineInstance> elements in the configuration).
• **-host host and -port port**: Send the shutdown single only to the engine instance running on host at port.

• **-force <seconds>**: Instruct the engine instance(s) being sent the shutdown signal to only wait <seconds> before forcibly terminating the instance.

⚠️ **Warning**

- **force** should only be used in the case in which an engine instance is not responding, as it may forcibly terminate running threads and prevent a clean shutdown.

**getstate**

The following script will return the state of one or more engine instances:

```bash
getstate [-all] [-host host] [-port port]
```

Engines may be in a 'running', 'paused', or 'offline' state.

Options:

• **-all**: Get the state of all engine instances identified by an `<EngineInstance>` element in the configuration.

• **-host host and -port port**: Get the state of the engine instance at host on port.

**pause**

The following script will instruct one or more engine instances to pause, suspending all activity and entering a wait state until either resumed or stopped:

```bash
pause -password <password> [-all] [-host host] [-port port]
```

You can pause engine nodes prior to performing database maintenance, then resume them when the database is back online.

Options:

• **-password <password>**: Required option, the administrative password for the engine instances.

• **-all**: Instructs the console to send a pause signal to all known engines (all engines listed in the `<EngineInstance>` elements in the configuration).

• **-host host and -port port**: Pause only the engine instance running on host at port.
**resume**

The following script will instruct one or more engine instances to resume, restoring all operations from the pause state:

`resume -password <password> [-all] [-host host] [-port port]`

Options:

- `-password <password>`: Required option, the administrative password for the engine instances.
- `-all`: Instructs the console to send a resume signal to all known engines (all engines listed in the `<EngineInstance>` elements in the configuration).
- `-host host` and `-port port`: Resume only the engine instance running on host at port.
The Workflow Engine is responsible for all operations that alter or advance the state of a workflow, including starting/stopping workflows, completing workitems, starting the next workitem(s) in the flow, and storing context parameters.

Workflow Engine

The engine is designed to perform the majority of this work asynchronously in the background to minimize impact on a user's experience. For example, from a functional standpoint, when a user completes a work item, the following must occur:

1. The workitem must be shifted into a completed state.

2. Any local context parameters the user has set on the workitem (such as key blocks on a Banner form) must be copied into the workflow context according to the mapping that the analyst setup for the workflow.

3. Subsequent activities in the workflow must be identified, transition rules evaluated, and the next workitem(s) must be started, given initial context values, and routed to the appropriate users’ worklists.

4. Email notifications for newly started workitems must be sent out.

To minimize the amount of time the user has to wait after submitting an item for completion, the engine simply shifts the item to a completed state, and internally queues up an event that this particular workflow needs further processing performed, then completes the transaction and returns control to the user. This completes step 1 above. A separate thread of execution in the engine then processes the event and performs steps 2-4 independent of any user interaction.
Therefore, for purposes of tuning, the engine can be thought of as consisting of two distinct parts:

- **Interactive**: Responsible for handling user requests to start/stop workflows, complete workitems, and set workitem parameters.

- **Asynchronous**: Responsible for performing the majority of work to advance workflows in the background.

**Note**

All tuning parameters are located in the `<EngineConfiguration>` elements under `<Engines>` in the configuration.xml file.

### Interactive

For the interactive parts, the only parameter of concern is the `<maxConnections>` element under `<JDBCPoolConfig>`. This controls the number of pooled JDBC connections the engine may use to service incoming requests to start/stop workflows, complete workitems, etc. In the example above, it's the number of connections it may use to complete step 1. This number represents the maximum number of concurrent requests to start a workflow, or change the state of an existing workflow (by completing an item, setting parameters, etc) that the engine can perform at once.

For example, if this number is set to 10, and 11 users simultaneously complete a workitem, then 10 of them will have their request go through immediately; the 11th will have to wait until one of the 10 requests is completed before being processed. These interactive requests are designed to be as small as possible, and are intended to execute very quickly, so in practice, a small pool should be able to service the requests of a large user base.

### Asynchronous

The asynchronous part of the engine is where the majority of the work is done. Asynchronous work is further divided into three distinct categories:

- core work
- external event processing
- notification work

There is a dedicated thread pool for each of the three types of work. Each thread is capable of processing a single unit of work at one time. In the example above, a thread from the main pool would complete steps 2-4 in a single transaction.
The asynchronous performance of the engine is controlled by the `<ThreadPools>` element. The `<Main>` element specifies the maximum and minimum number of threads the engine should keep on hand to perform work that advances workflows. If this number is set too low, then outstanding engine events may begin to build up. This will result in a delay in getting workitems onto users' worklists.

The `<ExternalEvent>` element controls the number of threads in the pool that is responsible for evaluating External Events and starting workflows in response to them. (See “External Events” on page 139 for more information.) If this number is too small, the engine may not be able to process External Events as fast as they are posted to the system. This will not cause any errors, or result in data loss, but can cause a delay between when the external system generates an event, and when a workflow is actually started for it.

Note

Each thread in the Main, Notification, and External Event pools is automatically assigned a JDBC connection that is independent of the pool configured in `<JDBCPoolConfig>`. This means that you can adjust all four values independently of one another.

## Engine Manager

The Engine Manager can be accessed from Workflow System Administration. The Engine Manager provides the ability to query the status and execute actions on the Workflow Engine(s) from within the Banner Workflow application. The Engine Manager is only available to those users who have a role that has been mapped to the `ug_admin_engineControl` security group.

The Engine Manager lists the host and port the engine is running on as described in the `configuration.xml` file. For more information on configuring the host and port please refer to “Update deployment information in `configuration.xml`” on page 24.
The status of engines can be Running, Idle and Stopped.

- **Running**: The engine is running and processing workflow events.
- **Idle**: The engine is running on the host and port but is currently not available to process events. Typically an engine is set to “Idle” when the database is unavailable. Once the database is available, the engine can be started without physically logging onto the Banner Workflow server and manually restarting the engine process.
- **Stopped**: The engine is not available on the host and port specified. Typically this is due to an administrator stopping an engine. When an engine is “stopped”, someone must log onto the Banner Workflow Server to restart the engine process.

**Note**
Stopping an engine from within Banner Workflow will require you to log onto the Banner Workflow Server in the future to restart the engine. It is not possible to start an engine from within the Banner Workflow application.

**Note**
All functions available within the Banner Workflow application are also available via the engineconsole script as detailed in “engineconsole” on page 161.

### How to Monitor Engine Performance

The `<JDBCPoolConfig>` can be the most difficult value to tune. In general, if users complain about a delay when performing operations such as starting workflows or completing workitems, you should try increasing this value. However, the delay may just as easily be caused by network problems, slow database access, or tuning problems in the Application Server. The `<JDBCConfigPool>` would be the first place to start to address this type of problem.

Monitoring the asynchronous threads is relatively straightforward, and can be accomplished by monitoring the ENG_EVENTS table.
**ENG_EVENTS Table**

In general, the number of rows in this table for which the column SCHEDULED_TIME is null should be zero or a small number. These rows denote pending internal engine events that should be processed immediately to advance a workflow, and thus the average number of such rows will tend to zero over any time period.

If you consistently see a large number of rows with a null SCHEDULED_TIME (several hundred), then this is an indication that the thread pools are not large enough to keep up with the throughput of the system. To determine which pool needs to be increased, look at the EVENT_TYPE column for rows in which SCHEDULE_TIME is null. If the event type begins with "immediate.notification", then the notification pool size should be increased. If the event type begins with "immediate.externalevent", then the External Event pool size should be increased. For any other types of event, the Main pool size should be increased. If you do not want to increase the size of the thread pool(s) for an engine instance, you can also start additional engine instances. User requests will automatically be forwarded among all available engine instances in a roughly "round robin" approach, thus spreading the workload among all the engine instances.

**Note**

If you want to be more detailed in your monitoring of this table, the CREATED_TIME column records the time at which the event was created. This time is represented as the specified number of milliseconds since the standard base time known as "the epoch", namely January 1, 1970, 00:00:00 GMT. To look for lagging events, you can run a query that returns all rows in ENG_EVENT for which SCHEDULE_TIME is null, and the CREATED_TIME is some specified distance in the past, for example 30 seconds. This would show you if there are any outstanding events that have been waiting more than 30 seconds to be processed.

You can also monitor these settings on a historical basis. A record of all successfully completed events is contained in the ENG_COMPLETED_EVENTS table, along with both the time the event was created (CREATED_TIME), and the time it was completed (COMPLETED_TIME). The difference between these two columns is the time in milliseconds it took the engine to complete processing of the event. Computing the difference for all events for which EVENT_TYPE starts with “immediate" will give you information on how long events which should be completed immediately, actually took to complete.

**Note**

There will always be some difference between the CREATED_TIME and the COMPLETED_TIME, but ideally it will be no more than a few seconds. Values larger than that can indicate that the pool sizes need to be increased.
**Note**

If you analyze the ENG_COMPLETED_EVENTS table, be sure only to look at rows where EVENT_TYPE begins with “immediate”. Rows with an event type beginning with “scheduled” indicate event scheduled to take place at a specified point in the future, and large differences between the CREATED_TIME and the COMPLETED_TIME are expected for these events and do not indicate a tuning problem.

**Note**

It is critical that production workflows be changed to an 'Active' state for maximum performance. Active workflows allow the engine to cache frequently used data. 'Test' mode workflows cannot be cached, and therefore can incur a significant performance penalty, both in terms of execution speed and the number of required database reads.
This chapter describes solutions to some of the common problems you may encounter when you use Banner Workflow with Banner.

**Launching Banner from Banner Workflow**

I am unable to launch Banner from Banner Workflow. I receive the following message:

*ERROR* You did not connect successfully; exiting.

It is necessary to have the UTL_HTTP object in your database. This object can be created by running the CATPROC.SQL script that is sent by Oracle. It should be in your ORACLE_HOME/rdbms/admin directory. This script should be run by the user SYS (not SYSTEM).

Users can be using the system when this script is run, however the script could take a long time to complete. It is recommended that the script be run while users are inactive.

**Processing Workflows**

**Workflows triggered by events do not start.**

Check the external event pages in Banner Workflow and search for the event in question. If the event was posted to workflow, it will show up in the search, along with its status. If the event could not be evaluated, there will be an error message indicating why the failure occurred. If the event was successfully evaluated, but did not start the expected workflow(s), you can view the event history, which will show the business processes which were evaluated for a possible start, whether or not their guard conditions passed, and which workflow was started for the process. If the external event is not visible from workflow, then it has not been 'posted' to the workflow system. In the case of Banner events, check the Event Queue Record Maintenance Form (GOAEQRM) for the status of the event.
A Banner form does not behave as predicted when it is launched as a workflow activity.

Check the following:

1. Verify that the **Workflow Enabled** checkbox on the Installation Control Form (GUAINST) is selected. This enables communication between Banner and Banner Workflow for your site.

2. Verify that your forms path points to the appropriate set of forms, including the most up-to-date version of the Banner Workflow-Awareness Library (GOQWFLW).

3. Determine whether the form complies with the guidelines for Workflow-enabled forms that can be found in “Handling Custom Forms” on page 61.

   If a baseline form does not comply with the guidelines, submit a contact to the ActionLine so the form can be corrected and made available to you. If a custom form does not comply with the guidelines, use the instructions contained in “Handling Custom Forms” on page 61, to modify the form as necessary.

4. If the form appears to comply with the Workflow-enabling guidelines, use one or both of the following methods to debug your form at runtime:

   - Use the instructions in the Processing Banner with Banner Workflow Chapter of Functional Integration Guide to add the **Workflow Debug** button to your custom toolbar. Relaunch the workflow activity. When the problem form appears, select the **Workflow Debug** button. Review the output to locate the problem.

   - For example, if the output does not list the form's associated parameters, the parameters may not have been extracted from Banner Workflow. Possibly, a **PRE-FORM** trigger in the form is preventing the **PRE-FORM** trigger in GOQOLIB from executing the parameter extraction logic of GOQWFLW.

   - If you locate a problem in a baseline form, report the problem to the ActionLine. If you locate a problem in a custom form, make the appropriate changes so the form complies with Banner standards. If you are unsure about the meaning of the debugging output, you can submit a contact to ActionLine, including the contents of the output.

   - Use the tracing capabilities of Banner to locate the problem.

   - The GOQRPLS library includes a packaged procedure that enables a Banner Oracle Forms module to spool information about its processing to a flat file (b2ktrace.log) in the user's TEMP directory. Information is spooled as long as the current environment is a Windows 32-bit operating system, the user's TEMP directory can be identified, and the value of the global variable `GLOBAL.DO_TRIGGER_TRACE` is set to `Y`. 
• By default, the trace variable is set to \( N \). To set it to \( Y \), ask your system administrator to enable you to access, in your `FORMS45_PATH`, a temporary copy of the executable for the GUAINIT Form that includes the following statement at the beginning of its `WHEN-NEW-FORM-INSTANCE` trigger:

\[
\text{COPY('Y', 'GLOBAL.DO_TRIGGER_TRACE');}
\]

• Then exit any Banner sessions that were opened by Banner Workflow and restart the appropriate workflow activity. When Banner Workflow reopens Banner, any triggers or subprograms within the Oracle Forms modules of Banner that spool information about their processing will be captured in a `b2ktrace.log` file in your TEMP directory. Review this file to determine what logic, if any, is failing to execute.

• For example, if the `b2ktrace.log` file doesn't contain the statement

\[
\text{G$\_WF\_CONTROL\_FORM.WF\_CONTROL\_NEW\_BLOCK\_BEGIN}'\]

for the execution of your form, then this packaged procedure has not been called in your form. This is most likely a result of a local `WHEN-NEW-BLOCK-INSTANCE` trigger in your form that is overriding the execution of the same trigger in GOQOLIB (which makes the above procedure call).

• If you locate a problem in a baseline form, report the problem to the ActionLine. If you locate a problem in a custom form, make the appropriate changes so it complies with Banner standards. If you are unsure about the meaning of the tracing output, you can submit a contact to the ActionLine, including the contents of the output.

### Missing required option - database error message appears

You tried to use an automated activity that does not have a database instance identifier correctly attached for the Banner product type. Attach the Banner product type.

### Processing Business Events

The status of a business event has been Ready for Processing for a long time.

The Event Dispatcher is probably down. Check with your system administrator.

### Customizing Example Workflows

The import fails when you try to move a customized example workflow into the production environment.

Before you start customizing an example workflow in your testing environment, you must initially import the same example workflow seed data into both your testing and production environments. The production environment must have all the data needed to support the new version of the example workflow.
Security Management in Banner Workflow

To grant users access to various sections of the Banner Workflow application, you need to assign roles to the various security groups in Security Management.

Each security group authorizes a role’s access to a particular area of Banner Workflow functionality.

Note

Only users in ug_admin_securityGroups may map roles to security groups.

Add a Role Authorization to an Existing Security Group

1. Click Workflow System Administration in Administration.
2. Click Security Management on the Workflow System Administration page.
3. Click the security group name that you want to add a role authorization for.
5. Select a role from the drop down list.
6. Click Save Authorized Role.
Delete a Role Authorization from an Existing Security Group

1. Click Workflow System Administration in Administration.

2. Click Security Management on the Workflow System Administration page.

3. Click the security group name that you want to delete a role authorization from.

4. Click the checkbox next to the authorized roles that you wish to delete.

5. Click Delete Selected Role Authorizations.

6. When you are asked to confirm the deletion, click OK.

Security Groups

Security groups are used to grant access to certain areas in Banner Workflow. Each group will have a corresponding role or roles assigned to it. Any user who has the role assigned to them will also be granted access to the particular security group area.

- **ug_admin**: Authorization to perform most administrative functions.
- **ug_admin_all_enterprise**: Authorization to perform all Enterprise related functions (event and process management, and related mapping functions necessary to setup external event processing).
- **ug_admin_bcc**: Authorization to create, update and delete reusable business components that can be used in workflow definitions.
- **ug_admin_calendar**: Authorization to create, update and delete work calendars that contain daily work hours for an office or institution.
- **ug_admin_engineControl**: Authorization to view and manage engine nodes.
- **ug_admin_enterprise**: Authorization to create, update and delete business processes and view the relationships between business processes, business events and workflow definitions.
<table>
<thead>
<tr>
<th>Role Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ug_admin_event_to_processes</td>
<td>Authorization to associate an event to a business process.</td>
</tr>
<tr>
<td>ug_admin_events</td>
<td>Authorization to create, update and delete business event definitions and event parameters.</td>
</tr>
<tr>
<td>ug_admin_externalEvents</td>
<td>Authorization to view external event logs and evaluate and process alerts for events that originate outside of Banner Workflow.</td>
</tr>
<tr>
<td>ug_admin_externalEvents_wparams</td>
<td>Authorization to view external event parameters and their associated values.</td>
</tr>
<tr>
<td>ug_admin_inprocess</td>
<td>Authorization to monitor metrics for in process workflow definitions.</td>
</tr>
<tr>
<td>ug_admin_modeler</td>
<td>Authorization to create, update and delete workflow definitions in the workflow modeler.</td>
</tr>
<tr>
<td>ug_admin_modeler_read</td>
<td>Authorization to view workflow definitions in the modeler.</td>
</tr>
<tr>
<td>ug_admin_organizations</td>
<td>Authorization to create and update organizations within the workflow system.</td>
</tr>
<tr>
<td>ug_admin_onlineUsers</td>
<td>Authorization to view a list of users who are currently logged into the workflow system.</td>
</tr>
<tr>
<td>ug_admin_process_roles</td>
<td>Authorization to define a role at an organization that can initiate a workflow process.</td>
</tr>
<tr>
<td>ug_admin_process_to_process_definitions</td>
<td>Authorization to associate workflow definitions to a business process.</td>
</tr>
<tr>
<td>ug_admin_productTypes</td>
<td>Authorization to create, update and delete product types.</td>
</tr>
<tr>
<td>ug_admin_remote_services</td>
<td>Authorization to invoke remote services. Necessary to execute import/export functionality, etc. Should only be given to users with a need for import/export or similar remote functionality.</td>
</tr>
<tr>
<td>ug_admin_roles</td>
<td>Authorization to view, create and update workflow roles and assign users to roles.</td>
</tr>
<tr>
<td>ug_admin_securedRoles</td>
<td>Authorization to secure a role and restrict its access to users. Authorization to assign restricted roles to users.</td>
</tr>
</tbody>
</table>
ug_admin_securityGroups  Authorization to map security groups to workflow roles.

ug_admin_static_documents  Authorization to add and remove static documents in the modeler.

ug_admin_technologyTypes  Authorization to create, update and delete technology types.

ug_admin_tools  Authorization to delete workflows with any status as well as authorization to run a report that analyzes what Banner components have changed.

ug_admin_users_fullAccess  Authorization to full administration rights to view, create, update and delete user accounts.

ug_admin_users_view  Authorization to read only access to search and view user accounts.

ug_admin_webservices  Used for Luminis integration.
Banner Workflow entity relationship diagrams will be displayed below. Each ERD diagram shows tables and views that show primary keys, foreign keys, alternate keys, table columns, and their relationships.

- Primary keys are noted with a key picture:
- Foreign keys are noted with a “(FK)” following the table column name.
- Alternate keys are noted with a “(AK)” following the table column name.
- Verb phrases are shown in middle of the relationship line. They describe the relationship that you create between a parent and child entity. Many of them in our models are shown with this notation “FK_” and then following a constraint name.

The diagrams show relationships and cardinality information is represented in Integrated Definition for Information Modeling (IDEF1X) modeling notation. This notation uses the following symbols:
### Cardinality Symbols Used

- **One to zero, one, or more**
  - Identifying
  - Non-identifying

- **One to one or more (P)**
  - Identifying
  - Non-identifying

- **One to zero or more (Z)**
  - Identifying
  - Non-identifying

- **One to exactly (N)**
  - Identifying
  - Non-identifying
**Component Query**

### Component Query Table

- **ID**: NUMBER(38)
- **NAME**: VARCHAR2(255)
- **DESCRIPTION**: VARCHAR2(255)
- **RELEASE_ID**: VARCHAR2(255)
- **STATUS**: VARCHAR2(255)
- **SOURCE**: VARCHAR2(255)
- **PRODUCT_TYPE_NAME**: VARCHAR2(255)
- **TECHNOLOGY_TYPE_NAME**: VARCHAR2(255)
- **CATEGORY_NAME**: VARCHAR2(255)
- **TECHNOLOGY_TYPE_ID**: NUMBER(38)
- **CATEGORY_ID**: NUMBER(38)
- **PRODUCT_TYPE_ID**: NUMBER(38)

### Component Category Table

- **ID**: NUMBER(38)
- **NAME**: VARCHAR2(255)
- **DESCRIPTION**: VARCHAR2(255)
- **LOCK_TOKEN**: NUMBER(38)

### Technology Type Table

- **ID**: NUMBER(38)
- **NAME**: VARCHAR2(255)
- **DESCRIPTION**: VARCHAR2(255)
- **CLIENT_LAUNCH_SERVICE_NAME**: VARCHAR2(255)
- **WEB_LAUNCH_SERVICE_NAME**: VARCHAR2(255)
- **LOCK_TOKEN**: NUMBER(38)

### Product Type Table

- **ID**: NUMBER(38)
- **NAME**: VARCHAR2(255)
- **VERSION**: VARCHAR2(255)
- **DESCRIPTION**: VARCHAR2(255)
- **DATA_SOURCE_NAME**: VARCHAR2(255)
- **LOCK_TOKEN**: NUMBER(38)

### Component-Technology Type Table

- **FK_COMPONENT_TECHNOLOGY_TYPE**

### Component-CATEGORY Table

- **FK_COMPONENT_CATEGORY**

### Component-PRODUCT Type Table

- **FK_COMPONENT_PRODUCT_TYPE**
Event Query

EVENT_QUERY
  ID: e.ID
  NAME: e.NAME
  DESCRIPTION: e.DESCRIPTION
  PRODUCT_TYPE_NAME: p.NAME
  PRODUCT_TYPE_ID: e.PRODUCT_TYPE_ID

EVENT
  ID: NUMBER(38)
  NAME: VARCHAR2(255)
  DESCRIPTION: VARCHAR2(255)
  PRODUCT_TYPE_ID: NUMBER(38)
  LOCK_TOKEN: NUMBER(38)

PRODUCT_TYPE
  ID: NUMBER(38)
  NAME: VARCHAR2(255)
  VERSION: VARCHAR2(255)
  DESCRIPTION: VARCHAR2(255)
  DATA_SOURCE_NAME: VARCHAR2(255)
  LOCK_TOKEN: NUMBER(38)
External Events

**Entity Diagram**

- **eng_external_event**
  - ID: NUMBER(38)
  - EXTERNAL_ID: VARCHAR2(255)
  - EXTERNAL_SOURCE: VARCHAR2(255)
  - EVENT_NAME: VARCHAR2(255)
  - PRODUCT_TYPE_NAME: VARCHAR2(255)
  - CURRENT_STATE: VARCHAR2(32)
  - LAST_CHANGE_DATE: NUMBER(38)
  - ORIGIN: VARCHAR2(255)

- **ENG_EVENTS**
  - ID: NUMBER(38)
    - WF_ID: NUMBER(38)
    - WI_ID: NUMBER(38)
    - EE_ID: NUMBER(38)
    - EVENT_TYPE: VARCHAR2(255)
    - SCHEDULED_TIME: NUMBER(38)
    - CREATED_TIME: NUMBER(38)

- **ENG_COMPLETED_EVENTS**
  - ID: NUMBER(38)
    - WF_ID: NUMBER(38)
    - WI_ID: NUMBER(38)
    - EE_ID: NUMBER(38)
    - EVENT_TYPE: VARCHAR2(255)
    - CREATED_TIME: NUMBER(38)
    - COMPLETED_TIME: NUMBER(38)

- **ENG_FAILED_EVENTS**
  - ID: NUMBER(38)
    - WF_ID: NUMBER(38)
    - WI_ID: NUMBER(38)
    - EE_ID: NUMBER(38)
    - EVENT_TYPE: VARCHAR2(255)
    - CREATED_TIME: NUMBER(38)
    - COMPLETED_TIME: NUMBER(38)
    - REASON: VARCHAR2(4000)
    - EXCEPTION_CLASS: VARCHAR2(4000)
    - MSG: VARCHAR2(4000)

- **ENG_EXTERNAL_EVENT_ERROR**
  - ID: NUMBER(38)
    - EE_ID: NUMBER(38)
    - TYPE: VARCHAR(32)
    - EVENT_ID: NUMBER(38)
    - SYSTEM_ERROR_TRACE: CLOB
    - MISSING_PARAM_NAME: VARCHAR2(255)
    - PROCESS_ID: NUMBER(38)
    - PD_ID: NUMBER(38)
    - ORG_ID: NUMBER(38)

- **ENG_EXTERNAL_EVENT_PARAM**
  - EVENT_ID: NUMBER(38)
    - NAME: VARCHAR2(255)
    - IS_NULL: VARCHAR2(1)
    - TYPE: VARCHAR2(255)
    - SEQ: NUMBER(38)
    - VALUE: VARCHAR(255)

- **eng_ee_history**
  - EE_ID: NUMBER(38)
  - EVENT_ID: NUMBER(38)

- **eng_ee_history_process**
  - EE_ID: NUMBER(38)
  - PROCESS_ID: NUMBER(38)
  - ORG_ID: NUMBER(38)
  - ACTIVE: VARCHAR(1)
  - PASSED_GUARD: VARCHAR(1)
  - PD_PK: NUMBER(38)
  - WF_PK: NUMBER(38)

- **eng_external_event_error**
  - ID: NUMBER(38)
  - EXTERNAL_ID: VARCHAR2(255)
  - EXTERNAL_SOURCE: VARCHAR2(255)
  - EXTERNAL_DATE: NUMBER(38)
  - EVENT_NAME: VARCHAR2(255)
  - PRODUCT_TYPE_NAME: VARCHAR2(255)
  - CURRENT_STATE: VARCHAR2(32)
  - LAST_CHANGE_DATE: NUMBER(38)
  - ORIGIN: VARCHAR2(255)
### Process Query

**Product Type**
- **ID**: NUMBER(38)
- **NAME**: VARCHAR2(255)
- **VERSION**: VARCHAR2(255)
- **DESCRIPTION**: VARCHAR2(255)
- **DATA_SOURCE_NAME**: VARCHAR2(255)
- **LOCK_TOKEN**: NUMBER(38)

**Event**
- **ID**: NUMBER(38)
- **NAME**: VARCHAR2(255)
- **DESCRIPTION**: VARCHAR2(255)
- **PRODUCT_TYPE_ID**: NUMBER(38)
- **LOCK_TOKEN**: NUMBER(38)

**Event Process Association**
- **ID**: NUMBER(38)
- **PROCESS_ID**: NUMBER(38)
- **EVENT_ID**: NUMBER(38)
- **ORG_ID**: NUMBER(38)
- **GUARD_CONDITION**: BLOB

**Process**
- **ID**: NUMBER(38)
- **NAME**: VARCHAR2(255)
- **DESCRIPTION**: VARCHAR2(255)
- **STATUS**: VARCHAR2(38)
- **LOCK_TOKEN**: NUMBER(38)

**Process Query**
- **ID**: NUMBER(38)
- **NAME**: VARCHAR2(255)
- **DESCRIPTION**: VARCHAR2(255)
- **STATUS**: NUMBER(38)

**Process Definition**
- **ID**: NUMBER(38)
- **IS_EXECUTION_PLAN**: VARCHAR2(1)
- **ORG_ID**: NUMBER(38)
- **NAME**: VARCHAR2(255)
- **DESCRIPTION**: VARCHAR2(255)
- **VERSION**: NUMBER(38)
- **STATUS**: NUMBER(38)
- **IS_ASSOCIABLE**: VARCHAR2(1)
- **PROCESS_START_ID**: NUMBER(38)
- **BEST_PRACTICE**: VARCHAR2(1)
- **CONFIDENTIAL**: VARCHAR2(1)
- **OWNER**: NUMBER(38)
- **ADMINISTRATOR**: NUMBER(38)
- **METRICS_ENABLED**: VARCHAR2(1)
- **ESTIMATED_TIME**: NUMBER(38)
- **LAGGING_PERCENTAGE**: NUMBER(38)
- **WORKCALENDAR_ID**: NUMBER(38)
- **NEXT_NODE_KEY**: NUMBER(38)
- **NEXT_ACTIVITY_NUMBER**: NUMBER(38)
- **LOCK_TOKEN**: NUMBER(38)

**Product Type**
- **ID**: NUMBER(38)
- **NAME**: VARCHAR2(255)
- **VERSION**: VARCHAR2(255)
- **DESCRIPTION**: VARCHAR2(255)

**Product Process Association**
- **ID**: NUMBER(38)
- **PROCESS_ID**: NUMBER(38)
- **PROCESS_DEFINITION_ID**: NUMBER(38)
- **ORG_ID**: NUMBER(38)
- **PROCESS_DEF_IS_ASSOCIABLE**: VARCHAR2(1)
- **EFFECTIVE_FROM**: NUMBER(38)
- **EFFECTIVE_TO**: NUMBER(38)

**Process Definition**
- **ID**: NUMBER(38)
- **NAME**: VARCHAR2(255)
- **DESCRIPTION**: VARCHAR2(255)

**Process Query**
- **ID**: NUMBER(38)
- **NAME**: VARCHAR2(255)
- **DESCRIPTION**: VARCHAR2(255)
- **STATUS**: NUMBER(38)
Runtime Views

**Workitems**

- ID: ENG_WORKITEM.ID
- TYPE: ENG_WORKITEM.TYPE
- ORG_ID: ENG_WORKITEM.ORG_ID
- PD_ID: ENG_WORKITEM.PD_ID
- ID: ENG_WORKITEM.NODE_ID
- CURRENT_STATE: ENG_WORKITEM.CURRENT_STATE
- LAST_STATE: ENG_WORKITEM.LAST_STATE
- START_DATE: ENG_WORKITEM.START_DATE
- STOP_DATE: ENG_WORKITEM.STOP_DATE
- WORKLIST_OWNER: ENG_WORKITEM.WORKLIST_OWNER
- NAME: ENG_WORKITEM.NAME
- ROLE_ID: ENG_WORKITEM.ROLE_ID
- MANDATORY: ENG_WORKITEM.MANDATORY
- CONFIDENTIAL: ENG_WORKITEM.CONFIDENTIAL
- COMPONENT_ID: ENG_WORKITEM.COMPONENT_ID
- PERFORMER_ID: ENG_WORKITEM.PERFORMER_ID
- RESERVED_DATE: ENG_WORKITEM.RESERVED_DATE
- LAGGING_DATE: ENG_WORKITEM.LAGGING_DATE
- OVERDUE_DATE: ENG_WORKITEM.OVERDUE_DATE
- RESERVED_DATE: ENG_WORKITEM.RESERVED_DATE
- PERFORMER_ID: ENG_WORKITEM.PERFORMER_ID
- COMPONENT_ID: ENG_WORKITEM.COMPONENT_ID
- CONFIDENTIAL: ENG_WORKITEM.CONFIDENTIAL
- MANDATORY: ENG_WORKITEM.MANDATORY
- ROLE_ID: ENG_WORKITEM.ROLE_ID
- NAME: ENG_WORKITEM.NAME
- DIRECTED_TO_USER: ENG_WORKITEM.DIRECTED_TO_USER

**Workflow**

- ID: ENG_WORKFLOW.ID
- ORG_ID: ENG_WORKFLOW.ORG_ID
- PD_ID: ENG_WORKFLOW.PD_ID
- ID: ENG_WORKFLOW.NODE_ID
- OWNER_ROLE_ID: ENG_WORKFLOW.OWNER_ROLE_ID
- ORIGINATING_PROCESS_ID: ENG_WORKFLOW.ORIGINATING_PROCESS_ID
- ORIGINATING_EVENT_ID: ENG_WORKFLOW.ORIGINATING_EVENT_ID
- ORIGINATING_USER_ID: ENG_WORKFLOW.ORIGINATING_USER_ID
- RUNNING: ENG_WORKFLOW.RUNNING
- PRIORITY: ENG_WORKFLOW.PRIORITY
- STOP_DATE: ENG_WORKFLOW.STOP_DATE
- START_DATE: ENG_WORKFLOW.START_DATE
- LAST_STATE: ENG_WORKFLOW.LAST_STATE
- CURRENT_STATE: ENG_WORKFLOW.CURRENT_STATE
- NAME: ENG_WORKFLOW.NAME
- PD_ID: ENG_WORKFLOW.PD_ID
- ORG_ID: ENG_WORKFLOW.ORG_ID
- ID: ENG_WORKFLOW.ID

**Workflow Alerts**

- ID: eng_workflow_alert.ID
- TYPE: eng_workflow_alert.TYPE
- ORG_ID: eng_workflow_alert.ORG_ID
- PD_ID: eng_workflow_alert.PD_ID
- ID: eng_workflow_alert.WORKITEM_ID
- MSG_PARAM: eng_workflow_alert.msg_param
- DATE_TIME: eng_workflow_alert.datetime
- WORKFLOW_NAME: ENG_WORKFLOW.NAME
- WORKFLOW_OWNER: ENG_WORKFLOW.OWNER
- WORKFLOW_ID: ENG_WORKFLOW.ID
- WORKFLOW_ALERT:eng_workflow_alert

**Process Definition**

- ID: NUMBER(38)
- TYPE: VARCHAR2(30)
- ORG_ID: NUMBER(38)
- PD_ID: NUMBER(38)
- ID: NUMBER(38)
- NAME: VARCHAR2(255)
- DESCRIPTION: VARCHAR2(255)
- VERSION: NUMBER(38)
- STATUS: NUMBER(38)
- IS_ASSOCIABLE: VARCHAR2(1)
- IS_EXECUTION_PLAN: VARCHAR2(1)
- START_DATE: NUMBER(38)
- END_DATE: NUMBER(38)
- NEXT_ACTIVITY: NUMBER(38)
- NEXT_ACTIVITY_NAME: NUMBER(38)
- NEXT_TRANSITION: NUMBER(38)
- NEXT_TRANSITION_NAME: NUMBER(38)
- NEXT_TRANSITION_TYPE: NUMBER(38)
- LOCK_TOKEN: NUMBER(38)
- ESTIMATED_TIME: NUMBER(38)
- LAGGING_PERCENTAGE: NUMBER(38)
- WORK_CALENDAR_ID: NUMBER(38)

**Process**

- ID: NUMBER(38)
- NAME: VARCHAR2(255)
- DESCRIPTION: VARCHAR2(255)
- VERSION: NUMBER(38)
- STATUS: VARCHAR2(32)
- LOCK_TOKEN: NUMBER(38)

**Workflow Web Definition**

- ID: ENG_WORKFLOW.ID
- ORG_ID: ENG_WORKFLOW.ORG_ID
- PD_ID: ENG_WORKFLOW.PD_ID
- ID: ENG_WORKFLOW.NODE_ID
- NAME: ENG_WORKFLOW.NAME
- CURRENT_STATE: ENG_WORKFLOW.CURRENT_STATE
- LAST_STATE: ENG_WORKFLOW.LAST_STATE
- START_DATE: ENG_WORKFLOW.START_DATE
- STOP_DATE: ENG_WORKFLOW.STOP_DATE
- PRIORITY: ENG_WORKFLOW.PRIORITY
- RUNNING: ENG_WORKFLOW.RUNNING
- ORIGINATING_USER_ID: ENG_WORKFLOW.ORIGINATING_USER_ID
- ORIGINATING_PROCESS_ID: ENG_WORKFLOW.ORIGINATING_PROCESS_ID
- ORIGINATING_EVENT_ID: ENG_WORKFLOW.ORIGINATING_EVENT_ID
- OWNER_ROLE_ID: ENG_WORKFLOW.OWNER_ROLE_ID
- ADMIN_ROLE_ID: ENG_WORKFLOW.ADMIN_ROLE_ID
- REQUIRES_EXECUTION_PLAN: VARCHAR(1)
- LAGGING_DATE: ENG_WORKFLOW.LAGGING_DATE
- OVERDUE_DATE: ENG_WORKFLOW.OVERDUE_DATE
- LAGGING_DATE: ENG_WORKFLOW.LAGGING_DATE
- OVERDUE_DATE: ENG_WORKFLOW.OVERDUE_DATE
- LOCK_TOKEN: ENG_WORKFLOW.LOCK_TOKEN
- ORIGIN: ENG_WORKFLOW.ORIGIN
- DESCRIPTION: VARCHAR2(255)
- ID: NUMBER(38)
- ORGANIZATION_ID: NUMBER(38)
- LOCK_TOKEN: NUMBER(38)

**Process**

- ID: NUMBER(38)
- NAME: VARCHAR2(255)
- DESCRIPTION: VARCHAR2(255)
- VERSION: NUMBER(38)
- STATUS: NUMBER(38)
- IS_ASSOCIABLE: VARCHAR2(1)
- IS_EXECUTION_PLAN: VARCHAR2(1)
- START_DATE: NUMBER(38)
- END_DATE: NUMBER(38)
- NEXT_ACTIVITY: NUMBER(38)
- NEXT_ACTIVITY_NAME: NUMBER(38)
- NEXT_TRANSITION: NUMBER(38)
- NEXT_TRANSITION_NAME: NUMBER(38)
- NEXT_TRANSITION_TYPE: NUMBER(38)
- LOCK_TOKEN: NUMBER(38)
- ESTIMATED_TIME: NUMBER(38)
- LAGGING_PERCENTAGE: NUMBER(38)
- WORK_CALENDAR_ID: NUMBER(38)

**Process**

- ID: NUMBER(38)
- NAME: VARCHAR2(255)
- DESCRIPTION: VARCHAR2(255)
- VERSION: NUMBER(38)
- STATUS: NUMBER(38)
- IS_ASSOCIABLE: VARCHAR2(1)
- IS_EXECUTION_PLAN: VARCHAR2(1)
- START_DATE: NUMBER(38)
- END_DATE: NUMBER(38)
- NEXT_ACTIVITY: NUMBER(38)
- NEXT_ACTIVITY_NAME: NUMBER(38)
- NEXT_TRANSITION: NUMBER(38)
- NEXT_TRANSITION_NAME: NUMBER(38)
- NEXT_TRANSITION_TYPE: NUMBER(38)
- LOCK_TOKEN: NUMBER(38)
SSL with Oracle Application Server

The Oracle Application Server (OAS) http server is based on Apache. The SSL is handled via mod_ossl, which is Oracle's SSL implementation.

The Oracle SSL implementation uses SSL certificates stored in an Oracle wallet, and managed by the Oracle Wallet Manager. The following describes the process for enabling SSL for workflow deployed on OAS. This document also describes how to be your own Certificate Authority, should you choose to do so. The steps should be the same for a full OAS install.

Note

Oracle also has a tech note on installing SSL (184432.1) available on metalink.oracle.com.

1. Launch the Oracle Wallet Manager.

2. Select Wallet->New to create a new wallet.

3. Select Wallet->Save As, and save the wallet. You may wish to put it under the apache directory, e.g., <OASinstall>/Apache/Apache/sslwallet.

4. Select Operation->Add Certificate Request to create a certificate request. You should use the server's hostname as the common name for the certificate.

5. Select Wallet->Save.

6. Export the certificate request.

7. Take the request to your certificate authority.

8. When you get the signed certificate back, select Operation->Import User Certificate to import it. You may be prompted to import the CA certificate.

9. Select Wallet->Save and exit the Oracle Wallet Manager.

You now need to enable the http server to use the certificate. To do so, edit the <OAS>/Apache/Apache/conf/httpd.conf file, either manually or via the Enterprise Manager.
Find the SSLWallet line, and replace it with one pointing to the directory you saved the wallet you created above in. If you aren't operating on a Windows system with the wallet configured for auto login, follow Oracle's instructions to add a SSLWalletPassword line as well.

You now need to ensure SSL is enabled for the http server in the opmn.xml file. This is usually enabled by default in Oracle AS R3, but not however in R2.

1. Open the Oracle Application Server configuration file, opmn.xml, found in ORACLE_HOME/opmn/conf.

2. Find the HTTP_Server reference (<ias-component id="HTTP_Server">).

3. Set or verify that the start-mode is set to "ssl-enabled" (it is ssl-disabled by default).

For example:

```xml
<ias-component id="HTTP_Server">
<process-type id="HTTP_Server" module-id="OHS">
<module-data>
<category id="start-parameters">
<data id="start-mode" value="ssl-enabled"/>
</category>
</module-data>
</process-type>
</ias-component>
```

At this point, OAS can run a secure version of the web site, and will use the SSL certificate you imported.

4. Save the file.

5. Restart the application server or optional just the http server component individually (R2).
Technical Note: Using SSL with both Banner Workflow and Banner

Because of a technical issue with Oracle procedures making external https calls, it is sometimes necessary to open an http port in the application server for Banner to communicate back to Banner Workflow. Client access to the Banner Workflow http port can be blocked via a shared firewall between the client and the servers running Banner Workflow and Banner.

When using this approach, be sure to define the launch parameter “workflow_protocol” to be http, “workflow_url” to be the host name of the Banner Workflow server, “workflow_root” to be the web application name, and “workflow_port” to be the available HTTP port for the OC4J instance. The parameters allow Banner to communicate back to Banner Workflow.

Web Service Applications

Additional steps are required if you want to make Web Services calls over https or have other application such as Banner Workflow Channels use SSL. For the Web Services frameworks using Java, keystore that the VM uses must include the SSL cert used on the server.

For java clients invoking the Web Services over https, you may need to import SSL certificate you installed in the Oracle Wallet as a trusted certificate in the java cacerts keystore for each box that you wish to invoke the Banner Workflow Web Services over https from.

To import the certificate, load the Oracle Wallet manager, and export the SSL certificate to a file. Find the cacerts file that your Java runtime uses (for typical JRE version 5 installs, it should be in <java_home>/jre/lib/security). Import the SSL certificate using keytool:

```
keytool -import -trustcacerts -alias oas -file myssl.cert -keystore cacerts
```

where myssl.cert is the SSL certificate you exported from the Oracle Wallet.

Note

The default password for the cacerts keystore is usually 'changeit' or 'default' depending on your version of java.
When prompted to trust the certificate, answer 'yes'.

**Note**
If you get a java exception of 'untrusted certificate chain' from a java program, it means that the JSSE (Java Secure Socket Extension) built into your Java installation cannot find the server's SSL certificate in its cacerts file.

**Note**
To use SSL with Banner Workflow Channels in Luminis, use the Luminis checkssl tool to import the certificate from the application server that is running Banner Workflow.
Alter Web Session Timeouts

The default session timeouts for Banner Workflow is 30 minutes. To alter the session timeouts edit the `<WebSessionTimeout>` element under `<AdvancedDeployment>` in the configuration.xml file.

After making the timeout adjustment a new EAR file needs to be re-deployed. Execute “bin/wftool updateSystem” from WORKFLOW_HOME and then follow the instructions in “Re-deploying after configuration changes” on page 35 to re-deploy the EAR file.
D 3rd Party Software Licenses

3rd Party Software Licenses

Banner Workflow 8.0 makes use of:

- Castor (http://castor.exolab.org) is Copyright 1999-2004 © Intalio Inc., and others. All Rights Reserved (http://castor.exolab.org/license.html)

- DHTML Calendar (http://www.dynarch.com/projects/calendar) was developed by Dynarch.com and is distrubted under the GNU Lesser General Public License (http://www.gnu.org/licenses/lgpl.html)

- Jakarta Oro and Jakarta Regexp (http://jakarta.apache.org/oro/) is developed by the Apache Software Foundation (http://www.apache.org/) that are made available under the Apache Software License (http://www.opensource.org/licenses/apachepl.php)

- JDOM (http://www.jdom.org/) is Copyright (C) 2000-2003 Jason Hunter & Brett McLaughlin. All rights reserved.

- JUnit (http://www.junit.org/) is made available under the Common Public License (http://www.opensource.org/licenses/cpl.php)

- Kunststoff Look&Feel (http://www.incors.org/) is distributed under the GNU Lesser General Public License (http://www.gnu.org/licenses/lgpl.html)

- Uportal was developed by theJA-SIG Collaborative (http://www.ja-sig.org/) and is Copyright (c) 2000 The JA-SIG Collaborative. All rights reserved.
