Oracle® Communications Design Studio

System Administrator's Guide

Release 7.4

F10638-01

October 2018

ORACLE®

Oracle Communications Design Studio System Administrator's Guide, Release 7.4

F10638-01

Contents

Preface

This guide contains information about administering Oracle Communications Design Studio. This guide includes information about configuring deployment settings for test environments, backing up and restoring Design Studio data, and automating builds.

Audience

This guide is intended for system administrators and other individuals who are responsible for ensuring that Design Studio is operating in the manner required for your business.

Related Documents

For more information, see the following documents in the Design Studio documentation set:

- *Design Studio Installation Guide*: Describes the requirements and procedures for installing Design Studio.
- *Design Studio Concepts*: Explains how to use Design Studio to manage and configure data for use across Oracle Communications service fulfillment products. This guide provides a conceptual understanding of Design Studio.
- *Design Studio Developer's Guide*: Provides an overview of Design Studio platform tools, and information about working with design patterns, externally created schemas, and source control. Finally, it provides information about deploying to production environments.
- *Design Studio Security Guide*: Provides an overview of security considerations, information about performing a secure installation, and information about implementing security measures in Design Studio.
- Design Studio Help: Provides step-by-step instructions for tasks you perform in Design Studio.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit

http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Document Revision History

The following table lists the revision history for this guide:

Version	Date	Description
F10638-01	October 2018	Initial release.

1

Configuring Deployment Settings

This chapter provides information about configuring deployment settings. You configure settings in Oracle WebLogic to ensure that individual Oracle Communications Design Studio users can deploy to run-time environments, and to enable SSL (secure sockets layer) on the WebLogic server. Additionally, you can define settings that determine how the deployment messages appear in the Console view.

Note: The procedures described in this chapter assume that you are using the WebLogic security realm. See the Oracle WebLogic Server documentation for installation and configuration instructions.

Setting Up Users for Design Studio Deployment

Before individual Design Studio users can deploy cartridges to run-time environments, you must create users (if necessary) and assign them to the WebLogic **Cartridge_Management_WebService** parent group. The run-time environment and the type of cartridges individual Design Studio users deploy determine to which additional parent groups users must be assigned.

To set up users for Design Studio deployment:

1. Log in to the WebLogic Administration Console.
2. Click **Security Realms**.

   The Summary of Security Realms page appears.
3. In the Realms table, click **myrealm**.
4. In **Settings for myrealm**, click the **Users and Groups** tab.
5. On the **Users** tab, click **New**.

   The Create a New User page appears.
6. In the **Name** field, enter the user name.
7. In the **Description** field, enter information about the user.
8. In the **Password** field, enter a password for the user.
9. Confirm the password, then click **OK**.

   The user appears in the Users table.
10. In the Users table **Name** column, click the user.

    The Settings page appears.
11. Click the **Groups** tab.

12. In the Parent Groups area, select **Cartridge_Management_WebService** from the **Available** column and move it to the **Chosen** column.
13. Click **Save**.

Enabling SSL Connections

Before individual Design Studio users deploy cartridges from Design Studio using an SSL connection, you must enable SSL in the WebLogic server to ensure that the Cartridge Management web service accepts the SSL connection. Additionally, you must define the environment connection parameters using HTTPS and include the correct SSL listening port.

> **Note:** When configuring SSL for WebLogic Server, define the minimum protocol version as Transport Layer Security (TLS) version 1.0. See *Oracle Fusion Middleware Securing Oracle WebLogic Server* for more information about configuring SSL.

The SSL keys must be made available to Design Studio and the keystore must include keys for any environment connection using SSL. See the Design Studio Help for information about defining SSL properties on the Studio Environment editor **SSL** tab.

To enable SSL connections:

1. Log in to the WebLogic Administration Console.
2. In the Environment area, click **Servers**.

   The Summary of Servers page **Configuration** tab appears.
3. In the Servers table, click the appropriate server.

   The **Settings** tabs appear.
4. On the **Configuration** tab **General** subtab, select **SSL Listen Port Enabled**.
5. In the **SSL Listen Port** field, enter the SSL port number.
6. Click **Save**.
7. Start Design Studio.
8. From the **Studio** menu, select **Show Environment Perspective**.
9. In the **Environment** tab, double-click an environment.

   The environment opens in the Studio Environment editor.
10. Click the **Connection** tab.
11. In the **Address** field, enter the following:

    https://*Host***:***Port***/cartridge/wsapi**

    where:

    Host is the host name of the system and *Port* is the SSL listening port number.
12. Click the **SSL** tab.
13. In the **Keystore** field, enter the location of the target server keystore.

    For example, in a test environment, you can enter the location for the **DemoTrust.jks** keystore (from the server installation). Or, you can enter a replica.

Note: If you are using a WebLogic Server DemoTrust keystore, and if the Java version installed on the client machine is version 1.7u40 or later, you must change the RSA key size. See "Changing the Keysize Value" for more information.

For more information about configuring identity and trust for WebLogic Server, see "Configuring Identify and Trust" on the Oracle Help Center:

http://docs.oracle.com/cd/E23943_01/web.1111/e13707/identity_trust.htm

Do not use the **DemoTrust.jks** keystore in a production environment.

14. Click **Save**.

Changing the Keysize Value

If you are using a WebLogic Server DemoTrust keystore, and if the Java version installed on the client machine is version 1.7u40 or later, you must change the RSA key size from 1024 Mbs to 256 Mbs in the **java.security** file.

For more information about configuring identity and trust for WebLogic Server, see "Configuring Identify and Trust" on the Oracle Help Center:

http://docs.oracle.com/cd/E23943_01/web.1111/e13707/identity_trust.htm

To change the key size value:

1. On the client machine, open the **java.security** file, located in the home directory:

   JRE_Home/**lib/security/java.security**

2. Change the following line:

   ```
   jdk.certpath.disabledAlgorithms=MD2, RSA keySize < 1024
   ```

   To:

   ```
   jdk.certpath.disabledAlgorithms=MD2, RSA keySize < 256
   ```

3. Save the file.

Configuring Deployment Messages

You use log levels to configure deployment messages for the Console view and for log files. You can modify these log levels in an application server, for example, WebLogic server, using WLST commands.

Note: Design Studio 7.4 does not support Log4j for modifying log levels. Instead, use WLST commands to configure deployment messages.

For information about logging custom WLST commands, see *Oracle Fusion Middleware WLST Command Reference for Infrastructure Components*.

For information about using WLST commands, see *Oracle Fusion Middleware Administering Oracle Fusion Middleware*.

Note: Refer to the corresponding WLST documents based on your application runtime WebLogic version.

Example 1–1 Sample WLST Commands for Configuring Deployment Messages

```
/scratch/oracle/FMW12c/Oracle_Home/wlserver/common/bin/wlst.sh
Initializing WebLogic Scripting Tool (WLST) ...
Welcome to WebLogic Server Administration Scripting Shell
Type help() for help on available commands
wls:/RI/serverConfig> connect('<userid>','<password>','<host>:<port>')
Connecting to <host>:<port> with userid <userid> ...
Successfully connected to Admin Server "AdminServer" that belongs to domain
"suite_domain".
Warning: An insecure protocol was used to connect to the
server. To ensure on-the-wire security, the SSL port or
Admin port should be used instead.
wls:/suite_domain/serverConfig> listLogHandlers(target='AdminServer',
name='console-handler')
Handler Name: console-handler
type:   oracle.core.ojdl.logging.ConsoleHandler
wls:/suite_domain/serverConfig> configureLogHandler(name="console-handler",
level="TRACE:32")
Handler Name: console-handler
type:   oracle.core.ojdl.logging.ConsoleHandler
wls:/suite_domain/serverConfig> setLogLevel(target='AdminServer',
logger='oracle.communications.platform.cartridgemanagement', level='TRACE:32',
addLogger='1')
wls:/suite_domain/serverConfig>
getLogLevel(logger='oracle.communications.platform.cartridgemanagement',
target='AdminServer')
TRACE:32
```

2

Backing Up and Restoring Design Studio Data

This chapter contains information about backing up and restoring the Oracle Communications Design Studio application, Design Studio projects, and Design Studio archives. Additionally, it describes a typical backup and restore strategy.

Backing Up Design Studio

Backing up Design Studio requires the following:

- Back up the Design Studio application. See "Backing-up the Design Studio Application" for more information.
- Back up Design Studio Projects. See "Backing Up Design Studio Projects" for more information.
- Back up Design Studio run-time archives. See "Backing Up Design Studio Run-Time Archives" for more information.

Backing-up the Design Studio Application

To ensure that the Design Studio application is not lost, archive each individual component as well as the assembled application environment.

Recommended frequency: Once per Design Studio release

Backup type: Manual

Recommended tooling: ZIP or TAR file

The application resources include any file required to run the Design Studio user interface or command-line builds. Back up the following application components:

- Oracle Enterprise for Eclipse package
- Design Studio feature TAR files
- Complete Design Studio environment
- Design Studio build scripts and configuration files
- Source control system application

Back up the complete Design Studio environment in ZIP or TAR format, and use the same archive to distribute to individual Design Studio users. See *Design Studio Installation Guide* for more information about distributing preconfigured installations.

Backing Up Design Studio Projects

Back up Design Studio project data files with every revision. Oracle recommends that you back up project data files using source control. See "Using Source Control to Back Up Design Studio Projects" for more information.

Recommended frequency: Daily

Backup type: Automated

Recommended tooling: Source control system

Backup the **project** folder, except for the following resources:

> **Note:** Do not place the following resources under source control.

- **.metadata** workspace folder and subfolders
- Java **bin** folder and subfolders
- **cartridgeBuild** folder and subfolders
- **doc** folder and subfolders
- Unsealed cartridges in the **cartridgeBin** folder

You can generate the content of a Design Studio project **cartridgeBin** folder (in unsealed cartridges) at any time. However, Oracle recommends that you define a production deployment archive strategy for run-time artifacts that is separate from project backups. See *Design Studio Installation Guide* for more information.

Include the **cartridgeBin** folder (for sealed cartridges) in backups and source control because run-time artifacts are not regenerated during sealed cartridge builds.

Using Source Control to Back Up Design Studio Projects

Oracle recommends that you use a source control system for backing up Design Studio projects. Design Studio includes support for the Eclipse platform Team functions, which enable Design Studio to interact with source control systems when managing resource changes in your projects.

Oracle recommends that you automate source control system backup. Automating this function to run daily ensures that a recent copy of the system is available at all times. Automated backups used in conjunction with regular user check-ins minimize potential data loss. Source control systems include complete sets of revisions for all files and folders, so maintaining multiple versions (Oracle recommends one week of versions) of the source control system backup is sufficient for redundancy.

Eclipse includes support for the Concurrent Versions System (CVS) programming environment, and additional plug-ins are available for other source control systems. For information about using Eclipse for CVS source control, see the discussion about team programming with CVS in the Eclipse *Workbench User Guide*. For information about what to source control in Design Studio, see *Design Studio Developer's Guide*.

Backing Up Design Studio Run-Time Archives

Design Studio creates run-time archives whenever individual Design Studio users build projects. Back up run-time archives to ensure repeatable production deployments of release versions.

Recommended frequency: Every milestone cartridge build

Backup type: Manual or automated

Recommended tooling: None required

Oracle recommends that you back up the Design Studio run-time archives that are used in the production deployment process. To back up the run-time archives, retain a copy of the **cartridgeBin** folder content of each project following a full command-line build. You can automate this backup by incorporating it into Design Studio command-line build scripts.

> **Note:** Design Studio produces run-time archives during the build process. These archives are saved to the **cartridgeBin** folder of the corresponding project. Design Studio produces run-time archives only for projects that represent a deployable cartridge. Projects that do not represent deployable cartridges (for example, Model projects) do not produce run-time archives.

See "Automating Builds" for more information.

Restoring Design Studio Data

Restoring Design Studio requires the following:

- Determine which type of recovery to use. See "About Data Recovery Types" for more information.
- Select a general strategy for data recovery. See "About Data Recovery Strategies" for more information.
- Restore the Design Studio application. See "Restoring the Design Studio Application" for more information.
- Restore Design Studio projects. See "Restoring Design Studio Projects" for more information.
- Restore Design Studio run-time archives. See "Restoring Design Studio Run-Time Archives" for more information.

About Data Recovery Types

There are two primary forms of data recovery, file media recovery and complete recovery.

- Use file media recovery when your individual Design Studio users must recover from a lost file, a damaged file, or from accidental or unintended file changes. You can perform file media recovery using source control or user-managed backup and recovery.

  The first step in performing file media recovery is to manually restore the file by copying it from a backup. Once you restore a file from a backup, Oracle recommends that you refresh the file in Design Studio to ensure that the application is using the current file and not a cached version. If using a source control system, you can recover the file using the source control system synchronization functions.

- Use complete recovery to recover from accidental or unintended file changes. You can perform complete recovery whether you use source control or user-managed backup and recovery.

About Data Recovery Strategies

Your approach to data recovery depends on whether the failure is media failure or user error, and the nature of the specific failure.

Strategies for Responding to Media Failure

Returning a system to operation following media failure depends on your backup strategy. In all cases, maintaining copies on multiple physical devices is critical. Recovery from backups stored on the same physical device may be possible if media failure is localized to a file or folder of the file system. However, the best protection strategy ensures data is always available on multiple physical devices in different physical locations.

Because it is often difficult to determine the files affected, a complete recovery is generally the best strategy to recover from media failure.

Strategies for Responding to User Error

User error failures require one of the following responses:

- Re-import the deleted file if a suitable copy or a previous version of the file exists.
- Re-enter the lost data manually if a record of data exists.
- Discard local changes and return the project to a previous state using source control.

Your backup strategy determines the recovery options that are available to you. For example, if you have no source control system, you have limited choices for restoring content to a particular point-in-time. Perform file media recovery if you can determine the files affected by the user error. Otherwise, perform a complete recovery.

Restoring the Design Studio Application

You must restore the Design Studio application to recover from a corrupted installation. To restore the Design Studio application, you instruct individual Design Studio users to do one of the following:

- Obtain and unzip a new prepackaged installation (that you provide).
- Reinstall the application using backups of any previously downloaded archives.

See *Design Studio Installation Guide* for more information.

Restoring Design Studio Projects

You can recover from a loss in a Design Studio project using file media recovery or complete recovery. The method you use depends on whether you can identify the affected Design Studio data files.

Restoring Projects Using File Media Recovery

If the affected files are few and easily identified, use file media recovery. When restoring project data using file media recovery, you can instruct individual users to:

- View local history for Design Studio resources.

  The Eclipse Local History feature recovery is a form of file media recovery that employs the use of the application's file history feature. The application can be configured to maintain copies of previous versions of a file. The local history of these files can be examined and used to replace or correct a current version. Source control implementations also include file versions committed to the repository to

provide a complete file change history. For more information about the CVS History view, see the Eclipse *Workbench User Guide*.

- Replace versions with local history or source control revisions.

  You can replace the current version of a resource with a revision from local history or source control using the CVS History view. See the Eclipse *Workbench User Guide* for more information about replacing a resource with local history.

- Restore deleted files.

  See the Eclipse *Workbench User Guide* for more information about restoring deleted resources from local history.

Restoring Projects Using Complete Recovery

Prior to performing a complete recovery, make a local backup of the corrupted project files. You may require this backup to recover content that was not present in the backup and is not easily reproducible.

Perform a complete recovery in a clean file structure (and not in an existing project) to ensure that old files are not included with the recovered content. If erroneous content can be isolated to specific projects, then recover those projects in their entirety. Otherwise, recover all projects in a new workspace.

When restoring project data using complete recovery, you can:

- Restore project data from a source control repository.

  The steps to recover a project from a source control repository vary by source control provider. For information about recovering projects when using CVS, see the Eclipse *Workbench User Guide*.

- Restore project data from a local backup.

  You can instruct individual users to restore Design Studio projects from a local backup. This approach may be required if you are not using a source control system or if the project requiring recovery has not been committed to source control but a local backup has been made.

  The local backup can be in archive format (ZIP or TAR) or can be a copy of the project folder in a different location. See Design Studio Help for information about importing projects.

Restoring Workspaces

To restore a workspace, you can instruct individual users to:

- Delete the **.metadata** folder to clear an existing workspace. A new workspace is created the next time Design Studio is launched.
- Import workspace preferences to a new workspace if they were previously exported for backup. See Design Studio Help for information about retaining workspace preferences.

Restoring Source Control Systems

If the source control repository is compromised through user error or media failure, it may be necessary to revert the repository to a recent automated daily source control repository backup. Use the source control system documentation to recover the backup of the source control repository.

Check that recent commits are present in the recovered source control repository. The repository will not include any committed content since the backup was made.

Recover recent revisions from local history and reapply them to the repository to bring the repository up-to-date.

Restoring Design Studio Run-Time Archives

You can restore run-time archives by:

- Reverting to a backup of the archive
- Reproducing the archive

Reverting to a Run-Time Archive Backup

The backup of the run-time archive is an exact copy. To restore the lost file, copy the cartridge archive from the backup repository to replace a missing or corrupted archive.

Reproducing a Run-Time Archive

When a backup of the archive is not available, you must rebuild the archive using the Design Studio projects. Recover the project versions for the milestone build from the source control repository to provide a point-in-time view of the related projects. This point-in-time view can be identified by a repository tag indicating a branch, version, or date. The choice of tag to restore to is dependent on the repository branch strategy. Oracle recommends that the tag and branch strategy include version tagging for any milestone build.

After projects needed to produce the archive are restored from the source control repository, you can initiate a build to reproduce the cartridge archive.

See "Automating Builds" for more information.

Design Studio Backup Strategy: Example

Table 2–1 describes a typical backup strategy that you can use as an example when setting up your own backup strategy.

Table 2–1 Typical Design Studio Backup Strategy

Resource	Frequency	Details
Design Studio Components	Archive every Design Studio release	Copy each installation component to the backup file storage. ZIP the final installation and copy to the backup file storage.
Source Control System Components	Archive every Source Control System release	Copy each installation component to the backup file storage.
Design Studio Data Files	Back up continuously	Back up every revision with local history. Keep files for 28 days. Track every committed data file change using source control.
Command Line Build Scripts	Back up continuously	Manage build script modifications using source control.
Source Control Repository	Archive Daily	Automate backup of the source control repository to the backup file storage. Maintain 7 recent versions.

Table 2–1 (Cont.) Typical Design Studio Backup Strategy

Resource	Frequency	Details
Project Data Files	Archive every project milestone	Clean all projects. Export projects to an archive. Manually copy archive to the backup file storage.
Design Studio Run-time Archives	Archive every project milestone	Tag the build in the source control system. Branch if desired. Manually copy build archive to the backup file storage.
File Storage	Back up continuously	Automatically replicate file storage using RAID. Replicate storage to a physically separate location.

3

Automating Builds

This chapter contains information on how to configure automated builds (command-line builds) to script build processes of cartridge projects and cartridge packs. Automated builds require no user interface interaction to build cartridge projects.

To automate Oracle Communications Design Studio builds, you create a process that builds a cartridge project and use a build automation system to schedule that process to run.

About Automated Builds

You can automate build processes of cartridge projects so that official builds can be made available to organizations such as testing or operations. You run automated builds in the Design Studio development environment by using Apache Ant tasks. For information about Apache Ant and to obtain downloads, see the Apache website at:

`http://ant.apache.org/`

You can customize the Ant **build-studio.xml** and **build.xml** scripts to automate the builds. The **build-studio.xml** script defines the Ant targets that build the cartridge. The **build.xml** script launches Design Studio and initiates the targets in the **build-studio.xml** script.

Oracle recommends that you add a validation to the automated build scripts to verify that successful builds produce the product archive file.

You create a new workspace in *DesignStudio_Home*/**template/workspace** and define the Design Studio preferences based on your build environment. You must define the preferences using a workspace template. Also, disable the **Build Automatically** option, found under the Design Studio **Project** menu.

If the workspace setup varies from one build machine to another, you cannot check the template workspace into source control. Oracle recommends that you check the workspace template into a source control system and keep all build machines consistent for all environment paths. If this is not possible, you must install the workspace template to a location on the build machine and update the build infrastructure settings accordingly.

About the Design Studio Development Directory Structure

In the Design Studio development environment, you can run automated builds using Apache Ant tasks. Example 3–1 shows the standard recommended directory structure for Design Studio development, with descriptions of each directory and file. The

directory structure should be placed under source control (except for the build folder and exclusions to project subdirectories).

Example 3–1 Standard Directory Structure for Design Studio Development

```
BASEDIR
   +- build-ant.xml       (Ant script for launching a build from Ant)
   +- build.bat           (Batch file for building on Windows)
   +- build.sh            (Shell script for building on UNIX platforms)
   +- scripts             (Subdirectory for build scripts)
      +- build.xml        (Main script to launch Design Studio build)
      +- build.xml-studio (Solution specific Design Studio build script)
   +- template            (Subdirectory for build templates)
   +- build               (Subdirectory for build output created by build)
   +- reports             (Subdirectory for target location of generated reports
   +- report-design       (Subdirectory for report designs not installed in
                           Design Studio)
   +- projects            (Subdirectory for each Design Studio project)
      +- [PROJECT 1]      (Design Studio projects needed for the build)
      +- [PROJECT 2]
            .
            .
      +- [PROJECT n]
```

To start the Design Studio **build.xml** script (located in the **scripts** directory), use the **build-ant.xml**, the **build.bat**, or the **build.sh** file, based on your build environment.

You must edit these files to set paths that are specific to the build environment. In addition, you must edit the **build-studio.xml** Ant script to define the solution-specific Design Studio tasks needed to build the cartridge or solution.

When working with this directory structure:

- The **template** folder is a clean workspace with environment-specific build properties. This workspace is used exclusively for the automated build.
- The **build** folder is created during the automated build. You do not create this folder or add it to source control.
- Add your Design Studio projects to the **projects** folder.

> **Note:** Do not check Development workspaces into source control. Developers should manage their own workspaces.

About Automated Reporting

You can integrate report generation into an automated build system if you want to automatically generate reports that capture the data modeled in Design Studio. These reports can include detailed information about an implemented solution. For example, the reports can capture the name, type, description, and relationships of projects, entities, and data elements. You can facilitate information sharing and data reviews by sharing these reports among team members who may not have Design Studio installed locally, or who require information about the data model in document form.

Design Studio includes a set of reference report designs that provide a foundational set of capabilities. You can use these report designs as is or as a starting point for customizing your own reports. For example, you can customize the report designs for content, layout, or branding.

You can also develop your own report designs. To develop your own report designs for use in Design Studio, you must first install the Eclipse BIRT feature. See *Design Studio Installation Guide* for more information about installing the BIRT feature. See *Design Studio Developer's Guide* for more information about developing custom reports and packaging custom reports into features.

You use report generation in your automated build processes by including the **studio.generateReport** Ant task in the **build-studio.xml** file. See "About the studio.generateReport Ant Task" and Example 3–3, "Sample build-studio.xml File" for more information.

About Ant Tasks

Ant tasks provide instructions for the build tasks. A script is required to start Eclipse to run the Ant build script.

The following Ant tasks are bundled with Design Studio:

- **studio.importProject**, which imports cartridge projects. See "About the studio.importProject Ant Task" for more information.
- **studio.buildProject**, which builds cartridges. See "About the studio.buildProject Ant Task" for more information.
- **studio.generateReport**, which integrates report generation into an automated build system. See "About the studio.generateReport Ant Task" for more information.

> **Note:** The Design Studio Ant tasks run only in Design Studio. You cannot run Design Studio Ant tasks as stand-alone Ant processes.

About the studio.importProject Ant Task

You import an Eclipse project into a workspace using the **studio.importProject** Ant task.

The **studio.importProject** Ant task has one parameter, **ProjectLocation**, which specifies the directory where the project description file is located. This Ant task does not make a copy of the project, it only references the project in the existing location.

> **Note:** When using the **studio.importProject** Ant task to import a project into a workspace, Oracle recommends copying the project into a build staging area. The example scripts in this chapter illustrate this recommendation.

About the studio.buildProject Ant Task

You build a project from the workspace using the **studio.buildProject** Ant task. Using **studio.buildProject** is the same as performing a full, clean build on a project through the user interface. The build is successful if the cartridge project contains no error markers and the cartridge archive is successfully created in the **cartridgeBin** folder.

The **studio.buildProject** Ant task has one parameter, **ProjectName**, which is the name of the project to be built (it may also include a customer name or prefix). The project, and all dependent projects, must be imported into the workspace prior to calling the build task.

About the studio.generateReport Ant Task

You use the **studio.generateReport** Ant task to integrate report generation into an automated build system and to automatically generate reports that capture the data modeled in Design Studio.

The **studio.generateReport** Ant task takes the following parameters:

- **id**: Specifies the identifier of the report type when generating reports using a design that is installed in the Design Studio feature. For example, the ID for the Entity Summary report is:

  oracle.communications.studio.report.reference.entityListReport

  You must define a value for either the **id** parameter or for the **file** parameter, but not both. An error is generated if you define values for both parameters or if you fail to define parameters for either parameter. See the *Design Studio Developer's Guide* for a list of all delivered report design IDs.

- **file**: Specifies the location of the report design when generating reports using a design that is saved to a file system (that is, report designs not installed in Design Studio). For example, a customized report path may appear as:

  C:\DesignStudio\report\sample.rptdesign

  Report locations should not be environment-specific. Design your locations so that the path can be located using a consistent mechanism on different machines. For example, you might use an Ant property to locate the report design folder. Use a path relative to the **BASEDIR**, rather than an absolute path reference.

- **format**: Specifies the format type of the report output.

  Note: Applications associated with output formats do not need to be installed to generate a report. For example, Adobe Reader does not need to be installed on a build system to generate a report in PDF format. However, a corresponding viewer or editor must be installed on a system to view or edit a generated report.

  The following format types are supported (a report design may only support a subset of the format types listed below):

  - **doc** (Word document)
  - **docx** (Word document XML)
  - **html** (Web document)
  - **odp** (OpenOffice presentation)
  - **ods** (OpenOffice spread sheet)
  - **odt** (OpenOffice Text)
  - **pdf** (Portable Document Format)
  - **ppt** (Power Point)
  - **pptx** (Power Point XML)
  - **xls** (Excel spread sheet)
  - **xlsx** (Excel spread sheet XML)
  - **xml** (Extensible Markup Language)

- **content**: Specifies how the scope of the generated report is organized. You define the overall scope of the report content using the value that you define in this parameter with the value that you define in the **dependency** parameter.

  You can define this parameter with:

  - **project**, which defines the scope of the report based on project containment, without concern for entity relations. The root projects are defined by the fileset.
  - **entity**, which defines the scope of the report based on a traversal of entity relations, starting from the entities defined in the fileset. The content represents the starting point and extends across project boundaries based on the dependency setting.

- **dependency**: Specifies whether the generated report includes content from dependent projects or sealed projects. You can define this parameter as:

  - **all**: When generating report content by project, use this value to include in the report all content in the selected projects as well as all content in all dependent projects. When generating report content by entity, use this value to include in the report all related entities in the selected entity project, and related entities in all dependent projects.
  - **unsealed**: When generating report content by project, use this value to include in the report all content in the selected projects as well as all content in all unsealed dependent projects. When generating report content by entity, use this value to include in the report all related entities in the selected entity project, and related entities in all unsealed dependent projects. When you use this value, no content from sealed dependent projects is included in the report.
  - **none**: When generating report content by project, use this value to include in the report all content in the selected projects as well as all content in all unsealed dependent projects. When generating report content by entity, use this value to include in the report all related entities in the selected entity project, and related entities in all unsealed dependent projects. When you use this value, no content from sealed dependent projects is included in the report.

- **output**: Specifies where the report is to be saved. For example, you can define this parameter as **${report-dir}\MyProjectSummary.pdf**.

- **fileSet**: Specifies the Ant fileset that represents the project or entity root for the collection of report content. All Ant fileset attributes and options are available when defining the value of this element. See the Apache Ant Project website for more information about the attributes and options available for use in this element:

  https://ant.apache.org/

  Note: You can include Ant script functions that perform actions after reports are generated. For example, you can publish reports to a shared location or to website; you can email the report to a distribution list, and so forth. See Apache Ant Project website for more information.

Automating Design Studio Builds

You automate Design Studio builds to generate cartridge archives without manual intervention by a user.

To automate Design Studio builds:

1. Install Design Studio on a build machine.

   See "Using Oracle Linux or Solaris for Automated Builds", or see "Installing Oracle Enterprise Pack for Eclipse" in the *Design Studio Installation Guide* for more information.

2. Install Ant.

   See "Installing Ant" for more information.

3. Start Design Studio.

4. Disable the Eclipse Usage Data Collector feature by doing one of the following:

   - If the Usage Data Upload dialog box appears during Eclipse startup, select **Turn UDC feature off**.
   - From the **Windows** menu, select **Preferences**, and then select **Usage Data Collector**. In the Usage Data Collector area, deselect **Enable Capture**.

5. From the **Project** menu, disable the **Build Automatically** option.

6. Create a workspace template.

   A workspace template includes build preferences that are specific to an environment.

   a. Create a new workspace in *DesignStudio_Home***/template/workspace**.

   b. Configure all applicable Oracle WebLogic Server, Java, and SDK home directories.

      – If individual users will be working with OSM projects, those users must configure the Oracle WebLogic Server, Java, and SDK home directories. For information about defining these OSM preferences in Design Studio, see *OSM Developer's Guide*.

      – If individual users will be creating custom web services or extending the data model for UIM projects, those users must complete the required preliminary setup. See *UIM Web Services Developer's Guide* and *UIM Developer's Guide* for more information.

   After you create a workspace template, Design Studio adds a **.metadata** directory to *DesignStudio_Home***/template/workspace**.

7. Determine the location and name of the cartridges to build with an automated process.

   For example, a cartridge path and name can resemble the following:

   *DesignStudio_Home***/build/workspace**

   Note: Oracle recommends managing the cartridge project with a source control system and basing builds on views of projects.

8. Create a directory called *DesignStudio_Home***/scripts/**.

9. In the *DesignStudio_Home***/scripts/** directory, create an Ant script called **build-studio.xml**, which will define the targets that build the cartridge.

   Oracle recommends using a copy of the sample **build-studio.xml** file provided. See "Example: build-studio.xml File" for more information.

10. Create a script to start Ant tasks inside Design Studio.

    a. In the *DesignStudio_Home*/**scripts/** directory, create an Ant script called **build.xml**, which will start Design Studio and start the build targets defined in **build-studio.xml**.

       Oracle recommends using a copy of the sample **build-studio.xml** file provided. See "Example: build.xml File" for more information.

    b. In the *DesignStudio_Home* folder, create a batch file (or shell script) to run the **build.xml** Ant script.

       See "Example: build.bat File for Windows" or "Example: build.sh File for UNIX" as a starting point for writing the batch file. If you are incorporating the build into an existing Ant-based build framework, see "Example: build-ant.xml File for Executing in Ant".

11. Run the batch file from a command line, from within the directory where the batch file resides.

12. Integrate the execution of the batch file into the automated build system.

Installing Ant

Apache Ant is an open source software tool often used for automating application build processes. Ant uses XML to define *targets*, which are executable commands that perform a specific task. By default, the XML file is named **build.xml**.

To install Ant:

1. Navigate to the Apache website historical archive page:

   `http://archive.apache.org/dist/ant/binaries`

2. Scroll down and click the **apache-ant-1.9.2-bin.zip** link.

   The File Download window appears.

   Note: Oracle recommends that you use Ant version 1.9.2.

3. Click **Save**.

   The Save As window appears.

4. Navigate to a local working directory and click **Save**.

5. Navigate to the local working directory where the downloaded ZIP file is saved.

6. Extract the contents of the ZIP file to a designated directory.

   The **apache-ant-1.9.2-bin** directory is created by the extraction, and the contents of the ZIP file are placed within this directory.

Example: build.xml File

The **build.xml** script starts Design Studio and starts the targets defined in the **build-studio.xml** script.

Example 3–2 Sample build.xml File

```
<!-- Use this script to stage the build and launch the Design Studio build
process. There should be no need to modify this script.  The tasks that run in
this script do not execute in Design Studio and do not have access to the Design
```

```
Studio build tasks.

To include environment specific content, modify the ${BUILD_SCRIPT} content. This
script expects a workspace template in the ${BUILD_DIR}/template/workspace
directory. The workspace should be a newly created empty workspace with suitable
workspace properties to run a build.  Workspace properties that require
configuration may include installation specific paths (for example, a path to an
SDK).-->

<project default="build" basedir="..">
    <property environment="env"/>
    <property name="build-dir" value ="${BUILD_DIR}"/>
    <property name="staging-dir" value ="${build-dir}/build"/>
    <property name="template-dir" value ="${build-dir}/template"/>
    <property name="template-ws" value ="${template-dir}/workspace"/>
    <property name="build-ws" value ="${staging-dir}/workspace"/>
    <!-- The automation build log is used by OSM automation build scripts output
-->
    <property name="automation-build-log" value ="${build-ws}/automation.log"/>
    <!--
        Stage-Workspace cleans the build area and initializes the workspace with
the workspace template.
    -->
    <target name="stage-workspace">
        <echo message="Cleaning Staging Directory: ${staging-dir}"/>
        <delete dir="${staging-dir}" quiet="true"/>
        <mkdir dir="${staging-dir}"/>
        <echo message="Cleaning Build Workspace: ${staging-dir}"/>
        <delete dir="${build-ws}" quiet="true"/>
        <mkdir dir="${build-ws}"/>
        <echo message="Staging Build Workspace from Template: ${template-ws}"/>
        <copy todir="${build-ws}/.metadata">
            <fileset dir="${template-ws}/.metadata"/>
        </copy>
    </target>
    <pathconvert pathsep=" " property="EquinoxJarPath">
        <sort>
            <fileset dir="${env.ECLIPSE_HOME}/plugins">
                <include name="**/org.eclipse.equinox.launcher_*.jar"/>
            </fileset>
        </sort>
    </pathconvert>
    <macrodef name="eclipseAntRunner">
        <attribute name="antFile"/>
        <attribute name="target"/>
        <sequential>
            <exec executable="${env.JAVA_HOME}\jre\bin\java">
                <arg value="-XX:NewRatio=5"/>
                <arg value="-XX:+UseAdaptiveSizePolicy"/>
                <arg value="-XX:+UseParallelGC"/>
                <arg value="-XX:MaxPermSize=256M"/>
                <arg value="-Xms256m"/>
                <arg value="-Xmx1024m"/>
                <arg value=
"-Dstudio.provisioning.automation.logger=org.apache.tools.ant.DefaultLogger"/>
                <arg value=
"-Dstudio.provisioning.automation.log.file=${automation-build-log}"/>
                <arg value="-jar"/>
                <arg value="${EquinoxJarPath}"/>
                <arg value="-data"/>
```

```
                <arg value="${build-ws}"/>
                <arg value="-application"/>
                <arg value="org.eclipse.ant.ui.antRunner"/>
                <arg value="-file"/>
                <arg value="@{antFile}"/>
                <arg value="@{target}"/>
                <arg value="-consoleLog"/>
            </exec>
        </sequential>
    </macrodef>
    <!--
        Build executes the BUILD_SCRIPT Ant script in Design Studio.

        The build script provided should include tasks for invoking the import
        and build of Design Studio projects.  Tasks required to run
        in the Design Studio run-time must be included in the provided build
script.
    -->
    <target name="build" depends="stage-workspace">
        <eclipseAntRunner antFile="${BUILD_SCRIPT}" target="all"/>
    </target>
</project>
```

Note: See Design Studio Help for information about troubleshooting and resolving memory and performance issues.

Example: build-studio.xml File

The **build-studio.xml** script is started by the **build.xml** script, and it runs inside of Design Studio.

The **build-studio.xml** script stages, imports, and builds cartridges, and generates reports:

- Stage: The script copies required projects from the project directory into a staging area.
- Import: The script loads each staged project into the workspace.
- Build: The script initiates the build procedure to generate the cartridge archive file.
- Report: The script initiates report generation to produce reports in a desired format.

Before running this script, make the following changes:

- Update the **project-dir** property to point to the Design Studio project directory. Projects are copied from this location during staging of the build.
- Update the **build-dir** property to point to the directory to contain the build. When pointing to an existing directory, ensure that it is empty (the contents may be deleted as part of the build process).

Important: Do not use the **project-dir** property for the **build-dir** property. Files in the build directory are deleted and recreated from the **project-dir** property during the build process.

Oracle recommends defining the **project-dir** and **build-dir** properties using **${basedir}**. This method ensures that you are using the directory of the active build. Do not use relative paths.

- Add a call to the **stage** target for each project required for the build. Include an entry for each dependent project.
- Add a call to the **import** target for each of the staged projects. Import dependent projects first.
- Update the **build** target to specify the cartridge project name to be built as the project.name property.
- Add calls to the **generateReportById** target and to the **generateReportByFile** target to define all required reporting parameter values. Example 3–3 includes examples of parameter values. You must update these values to reflect the values defined in your environment. See the *Design Studio Developer's Guide* if you require the report IDs for delivered Design Studio reports.
- Edit the **generateReportById** target and the **generateReportByFile** target to meet specific business needs. For example, you might edit these calls to log different information or to change the fileset structure.
- Add Ant tasks for additional post-generation reporting actions. For example, you might add Ant tasks for moving files, deploying files to a web server, bundling files into an archive file, and so forth.

Example 3–3 Sample build-studio.xml File

```
<project name="studioBuild" basedir="." default="all">
    <!-- Configure the project-dir and build-dir properties based on the build
environment -->
    <property name="project-dir" value ="${basedir}\projects"/>
    <property name="build-dir" value ="${basedir}\build"/>
    <property name="report-dir" value ="${basedir}\report"/>
    <property name="staging-dir" value ="${build-dir}/build"/>
    <property name="staging-projects" value ="${staging-dir}/prj"/>
    <property name="report-design-dir" value="${basedir}\report-design"/>
    <target name="stageProject">
        <delete dir="${staging-projects}/${project.name}" quiet="true"/>
        <copy todir="${staging-projects}/${project.name}">
            <fileset dir="${project.source}/${project.name}"/>
        </copy>
    </target>
    <target name="stage">
        <!-- Add a call to stage-project for each project required to produce
             the build. When using a source control system, the projects will
             be located in the view. -->
        <antcall target="stageProject">
               <param name="project.name" value="ModelProject1"/>
               <param name="project.source" value="${project-dir}"/>
        </antcall>
        <antcall target="stageProject">
               <param name="project.name" value="ModelProject2"/>
               <param name="project.source" value="${project-dir}"/>
        </antcall>
        <antcall target="stageProject">
               <param name="project.name" value="ModelProject3"/>
               <param name="project.source" value="${project-dir}"/>
        </antcall>
    </target>
    <target name="import">
        <!-- Add importProject calls for each project required to produce
             the build -->
        <antcall target="importProject">
```

```
                <param name="project.name" value="ModelProject1"/>
            </antcall>
            <antcall target="importProject">
                <param name="project.name" value="ModelProject2"/>
            </antcall>
            <antcall target="importProject">
                <param name="project.name" value="ModelProject3"/>
            </antcall>
        </target>
        <target name="build">
            <!-- Add buildProject calls to specify the cartridge project names to
                 be built as the project.name property. -->
            <antcall target="buildProject">
                <param name="project.name" value="ModelProject1"/>
            </antcall>
            <antcall target="buildProject">
                <param name="project.name" value="ModelProject2"/>
            </antcall>
            <antcall target="buildProject">
                <param name="project.name" value="ModelProject3"/>
            </antcall>
        </target>
    <target name="report">
        <!-- Add generateReport calls to specify the required parameter values.
                 The following antcalls are examples. The parameter values listed in
                 the generateReportById and generateReportByFile targets must be
                 changed to reflect your environment. See the Design Studio
                 Developer's Guide if you require the report IDs for
                 delivered Design Studio reports.-->
        <antcall target="generateReportById">
            <param name="report.id" value=
            "oracle.communications.studio.report.reference.compEntityStdDetailReport"/>
            <param name="report.format" value="pdf"/>
            <param name="report.dependency" value="all"/>
            <param name="report.output" value="${report-dir}\CompEntityReport.pdf"/>
            <param name="report.content" value="project"/>
            <param name="report.fileSetDir" value="${staging-projects}"/>
            <param name="report.fileSet" value="**/*.ddCartridge"/>
        </antcall>
        <antcall target="generateReportById">
            <param name="report.id" value=
            "oracle.communications.studio.report.reference.projectSummaryReport"/>
            <param name="report.format" value="html"/>
            <param name="report.dependency" value="all"/>
            <param name="report.output" value="${report-dir}\ProjectSummary1.html"/>
            <param name="report.content" value="project"/>
            <param name="report.fileSetDir" value="${staging-projects}"/>
            <param name="report.fileSet" value="**/*.ddCartridge"/>
        </antcall>
        <antcall target="generateReportById">
            <param name="report.id" value=
            "oracle.communications.studio.report.reference.projectSummaryReport"/>
            <param name="report.format" value="pdf"/>
            <param name="report.dependency" value="all"/>
            <param name="report.output" value="${report-dir}\ProjectSummary1.pdf"/>
            <param name="report.content" value="project"/>
            <param name="report.fileSetDir" value="${staging-projects}"/>
            <param name="report.fileSet" value="**/*.ddCartridge"/>
        </antcall>
        <antcall target="generateReportByFile">
```

```
        <param name="report.file" value=
           "${report-design-dir}\MyProjectSumary.rptdesign"
        <param name="report.format" value="pdf"/>
        <param name="report.dependency" value="all"/>
        <param name="report.output" value="${report-dir}\MyProjectSummary.pdf"/>
        <param name="report.content" value="entity"/>
        <param name="report.fileSetDir" value="${staging-projects}"/>
        <param name="report.fileSet" value="**/*.ddCartridge"/>
    </antcall
</target>
<target name="importProject">
    <echo message="Importing project: ${staging-projects}/${project.name}"/>
    <studio.importProject projectLocation="${staging-projects}/${project.name}"/>
</target>
<target name="buildProject">
     <echo message="Building project: ${staging-projects}/${project.name}"/>
     <studio.buildProject projectName="${project.name}"/>
</target>
<target name="generateReportByFile">
   <echo message="Generating Report: ${report.output}"/>
   <echo message="Report file: ${report.file}"/>
   <echo message="Report format: ${report.format}"/>
   <echo message="Report dependency: ${report.dependency}"/>
   <echo message="Report output: ${report.output}"/>
   <echo message="Report content: ${report.content}"/>
   <echo message="Report file set directory: ${report.fileSetDir}"/>
   <echo message="Report file set: ${report.fileSet}"/>
   <echo message="Report fileset: "/>
   <fileset id="fileset" dir="${report.fileSetDir}" includes="${report.fileSet}"/>
   <pathconvert pathsep="${line.separator}  " property="fileSet" refid="fileset"/>
   <echo>  ${fileSet}</echo>

   <studio.generateReport
      file="${report.file}"
      format="${report.format}"
      dependency="${report.dependency}"
      output="${report.output}"
      content="${report.content}">
      <fileset dir="${report.fileSetDir}" includes="${report.fileSet}"/>
   </studio.generateReport>

</target>
<target name="generateReportById">
   <echo message="Generating Report: ${report.output}"/>
   <echo message="Report id: ${report.id}"/>
   <echo message="Report format: ${report.format}"/>
   <echo message="Report dependency: ${report.dependency}"/>
   <echo message="Report output: ${report.output}"/>
   <echo message="Report content: ${report.content}"/>
   <echo message="Report file set directory: ${report.fileSetDir}"/>
   <echo message="Report file set: ${report.fileSet}"/>
   <echo message="Report fileset: "/>
   <fileset id="fileset" dir="${report.fileSetDir}" includes="${report.fileSet}"/>
   <pathconvert pathsep="${line.separator}  " property="fileSet" refid="fileset"/>
   <echo>  ${fileSet}</echo>

   <studio.generateReport
      id="${report.id}"
      format="${report.format}"
      dependency="${report.dependency}"
```

```
      output="${report.output}"
      content="${report.content}">
      <fileset dir="${report.fileSetDir}" includes="${report.fileSet}"/>
   </studio.generateReport>

</target>
    <target name="all" depends ="stage, import, build, report"/>
</project>
```

Example: build.bat File for Windows

Use Example 3–4 as a starting point for writing the Windows batch file.

> **Note:** Modify the **JAVA_HOME**, **ECLIPSE_HOME**, and **ANT_HOME** variables, based on the environment.

Example 3–4 build.bat for Windows

```
echo off
setlocal
if "%ECLIPSE_HOME%" == "" set ECLIPSE_HOME=C:\myPath\oepe-12.1.1.0
if "%JAVA_HOME%" == "" set JAVA_HOME=C:\myPath\jdk1.8.0_40-64
if "%ANT_HOME%" == "" set ANT_HOME=C:\myPath\apache-ant_1.9.2
REM - The BUILD_SCRIPT contains the build specific Ant tasks which run during
REM - the Design Studio build.  Update this file as needed.
set BUILD_SCRIPT=scripts\build-studio.xml
echo on
%ANT_HOME%/bin/ant -file scripts\build.xml -DBUILD_SCRIPT=%BUILD_SCRIPT%
```

Example: build.sh File for UNIX

Use Example 3–5 as a starting point for writing the UNIX batch file.

Run the batch file from a command line from within the directory where the batch file is located. You should integrate the execution of the batch file into your automated build system.

> **Note:** Modify the **JAVA_HOME**, **ECLIPSE_HOME**, and **ANT_HOME** variables, based on the environment.
>
> Design Studio automated builds support Oracle Linux 64-bit and Solaris 64-bit. The UNIX system must support graphical user interfaces.

Example 3–5 build.sh for UNIX

```
#!/bin/ksh
export JAVA_HOME=/myPath/jdk1.8.0_40-64
export ECLIPSE_HOME=/myPath/oepe-12.1.2.1.1
ANT_HOME=/myPath/apache-ant-1.9.2
# The BUILD_SCRIPT contains the build specific Ant tasks which run during the
# Design Studio build. Update the content of this file as needed.
BUILD_SCRIPT=scripts/build-studio.xml
$ANT_HOME/bin/ant -file scripts/build.xml -DBUILD_SCRIPT=$BUILD_SCRIPT
```

Example: build-ant.xml File for Executing in Ant

Use the **build-ant.xml** sample file in Example 3–6 as a reference for invoking the **build.xml** Ant script when launching the file from an existing Ant script. Use this sample if you intend to incorporate your automated build into an existing Ant-based build framework.

Example 3–6 build-ant.xml

```
<?xml version="1.0"?>
<project default="studio-build" basedir=".">
    <property environment="env"/>
    <target name="studio-build">
        <!-- Check if ECLIPSE_HOME and JAVA_HOME environment properties
             are set -->
        <fail unless="env.ECLIPSE_HOME">
             ECLIPSE_HOME environment property must be set.</fail>
        <fail unless="env.JAVA_HOME">
             JAVA_HOME environment property must be set.</fail>
        <ant antfile="scripts/build.xml">
            <property name="BUILD_SCRIPT" value ="scripts/build-studio.xml"/>
        </ant>
    </target>
</project>
```

Using Oracle Linux or Solaris for Automated Builds

Design Studio supports Oracle Linux 64-bit and Solaris 64-bit for automated build capability only. If you are using Linux or Solaris for automating builds, you must configure proxy settings. Additionally, you must install the following:

- If you are using a Linux 64-bit system for automated builds, you must install the Eclipse 32-bit or the Eclipse 64-bit platform for Linux, as well as Java 32-bit or 64-bit support, as appropriate.
- If you are using a Solaris 64-bit system for automated builds, you must first install the Eclipse 32-bit platform for Solaris, as well as Java 32-bit support.

> **Note:** Oracle recommends using Windows for automated builds.

When automating builds using Oracle Linux and Solaris, see the following topics:

- Installing Oracle Enterprise Pack for Eclipse Linux
- Installing Eclipse for Solaris
- Configuring Proxy Settings

Installing Oracle Enterprise Pack for Eclipse Linux

To download and install Oracle Enterprise Pack for Eclipse Linux (64-bit):

1. Install and configure Oracle Enterprise Pack for Eclipse.

   See *Design Studio Installation Guide* for more information.

> **Note:** After accepting the license agreement, click the **Linux (64-bit)** link.

2. If using a proxy server, configure the proxy settings.

   See "Configuring Proxy Settings" for more information.

Installing Eclipse for Solaris

To download and install Eclipse for Solaris:

1. Navigate to the Eclipse download website:

   http://download.eclipse.org/eclipse/downloads/

   Note: Site layout, content, and procedures are subject to change based on updates by Eclipse.org administrators.

2. Click the link for the latest release.

   The Eclipse Release Build page appears.

3. Click the Eclipse SDK link for the appropriate platform.

   The Eclipse Downloads - mirror selection page appears.

4. Click a mirror site.

   The File Download dialog box appears.

5. Click **Save**, and then specify the location to save the ZIP file.

6. Click **Save** again, which begins the download process.

7. When the download completes, unzip the file and create a shortcut on the desktop to the **eclipse.exe** file.

   For example, unzip the file into a new folder called **DesignStudio**.

8. Do one of the following:

   - Install the Java Runtime Environment.

     Obtain the Java Runtime Environment from the Oracle website at:

     http://www.oracle.com/technetwork/java/javase/downloads/index.html

   - Locate an existing Java Runtime Environment installation.

     Java Runtime Environment typically resides in the **Program Files/Java** folder.

9. Copy the JRE folder (for example, **jre1.8.0_71**) into the Eclipse folder at the same location as the **eclipse.exe** file.

   Note: If Eclipse does not locate the JRE folder in the directory in which Eclipse was installed, it attempts to find the JRE using the JAVA_HOME windows environment variable. See *Design Studio Installation Guide* for more information about required JDK versions.

10. Rename the JRE folder to **jre**.

11. Configure the startup properties in the **eclipse.ini** file located in the **Eclipse** folder.

    See *Design Studio Installation Guide* for more information.

12. Open Eclipse.

13. If using a proxy server, configure the proxy settings.

    See "Configuring Proxy Settings" for more information.

14. Install additional required Eclipse features.

    See "Installing Additional Required Features" for more information.

Installing Additional Required Features

To use Solaris to run automated builds for Design Studio, you must install additional required Eclipse features.

To install additional required features:

1. Start Design Studio.
2. From the Design Studio **Help** menu, select **Install New Software**.

   The Available Software dialog box appears.
3. In the **Work with** field, select the **http://download.eclipse.org/releases/***version* option, where *version* is the name of the Eclipse version that you have installed.

   A list of folders and features appears.
4. Expand the **Modeling** folder and select the following features:
   - **EMF - Eclipse Modeling Framework SDK**
   - **Graphical Editing Framework Zest Visualization Toolkit SDK**
   - **XSD - XML Schema Definition SDK**
5. Expand the **Web, XML, Java EE and OSGi Enterprise Development** folder and select the following features:
   - **Eclipse XML Editors and Tools**
   - **Eclipse XSL Developer Tools**
   - **CXF Web services**
6. Deselect **Contact all update sites during install to find required software**.
7. Click **Next**.

   The Install Details dialog box appears.
8. Review the installation details.
9. Click **Next**.

   The Review Licenses dialog box appears.
10. Accept the license agreement.
11. Click **Finish**.
12. When prompted, click **Yes** to restart Eclipse.

Configuring Proxy Settings

If you are using a proxy server and intend to run automated builds using Oracle Linux or Solaris, you must configure the proxy server in Eclipse.

To configure the proxy server in Eclipse:

1. If you do not know the proxy address and port of the intranet proxy, do one of the following:

- If the proxy is statically configured, obtain the proxy address and port from the Internet browser LAN Connection settings. See your browser Help for information about obtaining the values in the Proxy Server Address and Port fields. Contact your network administrator if you can not access the Internet browser settings.
- If the proxy is not statically configured, acquire the settings from the automatic configuration script. If the proxy is not statically configured, no specific proxy address and port number entries are defined in the connection settings. Open the DAT file in a text editor to obtain the proxy address and port number. See your browser Help for information about locating the DAT file.

2. Start Eclipse.
3. From the **Window** menu, select **Preferences**, then select **General**, and then select **Network Connections**.
4. In the **Active Provider** field, select **Manual**.
5. In the Proxy Entries table, double-click the **HTTP** schema entry.

   The Edit Proxy Entry dialog box appears.
6. In the **Host** field, enter a host name.
7. In the **Proxy** field, enter the proxy number.
8. (Optional) To require user authentication, select **Requires Authentication**, then enter the user name and password required for access.
9. (Optional) In the Proxy Entries table, double-click the **HTTPS** schema entry.

   In the Edit Proxy Entry dialog box, enter the proxy name, port number, and user authentication information.
10. Click **OK**.
11. Click **OK** again, which closes the Preferences dialog box.