ellucian

Banner

Student Self-Service Installation Guide

Release 9.7
September 2017

Ellucian
2000 Edmund Halley Drive
Reston, VA 22033
United States of America

Contents

Introduction 8

Important changes 8

Use Ellucian Solution Manager to install your product upgrades 8

Install guide organization 9

Known installation issues 9

Hardware requirements 9

CPU and memory 9

Screen resolution 9

Browsers 10

Tablets and mobiles 10

Software requirements 10

Oracle Database 10

Application server 11

Middle Tier (application server) platforms 11

Developer Requirements 12

Ellucian software 12

Security setup 13

Single sign on (SSO) support 13

Supported browsers 14

Chrome Java Support 14

Chrome Java support on Mac OS X 14

Chrome Compatibility Mode 15

Internet Explorer Compatibility View 15

Java dependencies 15

Enable Application Navigator 16

Deployment of multiple web applications 16

With CAS or SAML2 SSO 16

Without SSO 17

F5 load balancer configuration 17

Upgrade the Database 18

Perform the Banner DB Upgrade steps 18

Update login.sql 18

Verify that the required products are applied 19

Verify the ban_ss_user database account 19

Set up access for application users with an administrative account 19

Migrate staged files to the permanent directories 19

Unix 20

Windows 21

Update the version number 21

Install Banner Student Self-Service Application for CAS SSO 22

Undeploy the existing application 22

Tomcat 23

Undeploy using the Tomcat Manager web application 23

Undeploy using a manual procedure 23

WebLogic 24

Customize the WAR file 25

Unzip the release package 25

Prepare the installer 25

Install into the product home directory 26

Configure shared settings 28

JNDI datasource 28

Link to Self-Service Banner 8.x 28

Navigational MEP support from Student Self-Service to Banner Self- Service 8.x . 29
Session timeouts 30
Number of supported email recipients 32
Configure footerFadeAwayTime 32
Location of photographs 32
Default photograph 33
Application properties 33
Configure application-specific settings 33
Cross-frame scripting vulnerability 34
Configure Extensibility in Student Self-Service application 34
Bookstore links 35
Theme Editor tool 36
Google Analytics 38
Quartz Scheduler Configuration Settings 39
Self-service end point 39
Integration with Ellucian Degree Works 39
Navigation to other applications 40
Configure View Grades page via SQL script 40
Configure Class List page via SQL script 40
Configure Drop Roster page via SQL script 41
JMX MBean name 41
Location of the logging file 41
Logging level 42
Proxied Oracle users 42

Setup CAS SSO Configuration 43

Authentication Provider Name 43

CAS SSO Configuration 44
CAS SSO Configuration 44
Logout URL 45
Password reset 46
Redirect pages in a MEP environment 46

Institutional Home Page Redirection Support 46

Regenerate the WAR file 47

Configure and deploy the WAR file to a web application server 48

Tomcat 48
Configure the Tomcat server 48
Configure Java Management Extensions 52
Deploy the WAR file to the Tomcat server 53

WebLogic 55
Verify WebLogic prerequisites 55
Set up the cookie path 55
Create a WebLogic machine 56
Create a WebLogic server 56

Configure weblogic.xml file to make Banner 9.x JSession cookie secure 57
Update Oracle JDBC JAR files on the WebLogic server 58
Create an administrative datasource and connection pool 59
Create a self-service datasource and connection pool 60
Configure server communication 61
Deploy and start the application in the WebLogic server 62

Configure the application 63

Name format 63

Phone format 63

Date format 64
default.date.format 64
js.datepicker.dateFormat 65
Time format 65

Multiple calendars 66

CSS customization 66

Institution name 67

Custom JavaScript 67

Install Banner Student Self-Service Application for SAML 2.0 SSO . . . 68

Undeploy the existing application 68

Tomcat 68
Undeploy using the Tomcat Manager web application 69
Undeploy using a manual procedure 69

WebLogic 70

Customize the WAR file 70

Unzip the release package 71

Prepare the installer 71

Install into the product home directory 72

Configure shared settings 73
JNDI datasource 73

Generate SAML 2.0 metadata files 74

Create a keystore (*.jks) file 74

Create a service provider file 74

Create an identity service provider file 76

Add an IDP Certificate entry in the newly created keystore file 77

Add keystore, service provider, and identity service provider files to WAR file creation location 77

Configure application-specific settings 77
Self-service end point 77
JMX MBean name 78
Location of the logging file 78
Logging level 78

Set up SAML 2.0 SSO Configuration 79
Authentication Provider Name 79
Logout URL 80
SAML 2.0 SSO Configuration 80

Customize the landing page background image 83

Regenerate the WAR file 85

Configure and deploy the WAR file to a web application server 86
Tomcat 86
Configure the Tomcat server 86
Configure Java Management Extensions 88
Deploy the WAR file to the Tomcat server 89
WebLogic 91
Verify WebLogic prerequisites 91
Create a WebLogic machine 91
Create a WebLogic server 91
Update Oracle JDBC JAR files on the WebLogic server 93
Create an administrative datasource and connection pool 94
Deploy and start the application in the WebLogic server 95

Add service provider in Ellucian Ethos Identity Server 96
Add SAML settings 96

Modify Identity Provider Issuer 98

SAML 2.0 Configuration Sub-tasks 99
Create a keystore (*.jks) file 99
Extract a X509 Certificate Key 100
Extract X509 certificate data 102
Add IDP certificate entry to the .jks file 103

Introduction

This installation guide details the steps that are required to install the components of Banner Student Self-Service.

Before you install any components of the system, you should review this chapter thoroughly so you have a better understanding of what you are installing and where you will install it.

Important changes

This release of Student Self-Service 9.7 delivers Drop Roster feature, details for which can be found in the Banner Student Self-Service Handbook 9.7 This also incorporates the changes delivered for the Student Self-Service 9.6.0.1 and Student Self-Service 9.6.0.2.

Use Ellucian Solution Manager to install your product upgrades

Ellucian recommends that you use Ellucian Solution Manager to perform Banner product upgrades, rather than using a manual installation process.

With Solution Manager, you can:

- Identify dependency information within and between products.
- View the latest version numbers for the Banner products you have installed, along with all other version numbers installed in your environment.
- Use the Get New Releases feature to identify available upgrades and download them immediately.
- Identify and install product pre-requisites, along with any upgrades you have selected.

Solution Manager currently supports most Banner 8 and 9 products. For more information on the Banner product versions currently supported by Solution Manager, see the *Banner Upgrades Support Status* guide. For detailed instructions on how to install and configure Solution Manager, see the latest *Solution Manager User Guide*.

Install guide organization

After you have reviewed the introduction and completed the database upgrade tasks in the "Upgrade the Database" chapter, you can install the applications for Banner Student Self-Service using CAS Single Sign On (SSO) or SAML 2.0 Single Sign On (SSO) protocols.

See the following chapter for installation with the CAS SSO protocol:

"Install Banner Student Self-Service Application for CAS SSO" on page 22

See the following chapter for installation with the SAML 2.0 SSO protocol:

"Install Banner Student Self-Service Application for SAML 2.0 SSO" on page 68

Known installation issues

Before you install Banner Student Self-Service, refer to the following articles on the Ellucian Support Center (http://www.ellucian.com/Solutions/Ellucian-Client-Support/) for any issues that were reported after the release was posted:

- Banner DB Upgrade 9.12 upgrade issues (banner-db-upgrade-91200u) - Article number - 000039616
- Banner Student Self-Service 9.7 upgrade issues - Article number - 000039898

Hardware requirements

The application has the following hardware requirements.

CPU and memory

Recommended:	Quad core CPU with 4 to 8 GB of memory for the application server
Minimum:	Dual core CPU with 2 GB of memory for the application server

Screen resolution

The minimum PC screen resolution for the application is 1024 x 768.

Browsers

Drop roster Self-Service pages were tested with browsers and other devices as per Ellucian's compatibility as follows:

- Internet Explorer 11
- Chrome 61.x
- Firefox 55.x

Tablets and mobiles

The application is supported on the following:

- Tablets
  - iPad iOS 10.2.1
- Mobiles
  - iPhone, Android
- Android
  - OS 7.0 & 7.1.1
- Windows Surface
  - Windows 8.1

Software requirements

The application has the following software requirements.

Oracle Database

Supported versions of the Oracle Database depend on multiple factors, including third-party support time lines. For a complete list of supported Oracle technologies, refer to the

Ellucian Oracle Support Calendar. The calendar is available in the *Interactive Banner Compatibility Guide* on the Ellucian Download Center.

Application server

The application is supported on the following application servers:

- Oracle WebLogic 10.3.3, 10.3.4, 10.3.5, and 10.3.6, and Weblogic 12.1.3
- Apache Tomcat 7 and 8

Oracle Fusion Middleware (OFM) consists of several software products, including WebLogic Server. WebLogic Server is required for an Oracle Banner 9.x application server environment.

No other OFM products are required. However, if an SSL-enabled Oracle HTTP Server (OHS) port is used, the Oracle Web Tier should also be installed in order to use the `mod_wl_ohs`.

Middle Tier (application server) platforms

The application is supported on the following application server and operating system combinations:

Tomcat (64 bit)	WebLogic (64 bit)
Red Hat Linux 6.x	Red Hat Linux 6.x
Windows Server 2008	Windows Server 2008
Solaris 10	Solaris 10
AIX 6.1 (JDK 1.7 SR10 or later)	AIX 6.1 (JDK 1.7.0 SR10 or later)
HP-UX	HP-UX 11iV3 (11.31)

Note: Banner 9.x applications were tested on WebLogic using both the Classic Domain template and the Basic Domain template.

For WebLogic server environments, JPA 2.0 support must be enabled. WebLogic server does not enable JPA by default. To enable JPA, use the steps in the appropriate Oracle documentation:
WebLogic 10.3.3:
http://docs.oracle.com/cd/E14571_01/web.1111/e13720/using_toplink.htm#i1221315

WebLogic 10.3.4:
http://docs.oracle.com/cd/E17904_01/web.1111/e13720/using_toplink.htm#i1221315

WebLogic 10.3.5:
http://docs.oracle.com/cd/E21764_01/web.1111/e13720/using_toplink.htm#EJBAD1309

WebLogic 10.3.6:
http://docs.oracle.com/cd/E23943_01/web.1111/e13720/using_toplink.htm#autold2

Developer Requirements

Developers who pull the source code from GIT will need to use the following versions of software when making extensions.

- Grails 2.5
- JDK 1.7
- Application servers
  - Oracle Middleware
    - Using WebLogic 10.3.3, 10.3.4, 10.3.5, 10.3.6 and 12.1.3.0.0
    - Apache Tomcat 7 and 8
- Oracle 11.2.0.4 or higher
- Java – 7 and 8
- CPC version – 9.1
- Release GIT Tag for Student: rel-studentselfservice-9.7

Ellucian software

The following product upgrades must be applied:

- Banner Database Upgrade 9.12
- Banner General 8.9.2 with patch pcr-000150752_gen8090201
- Banner Student 8.13.2.1 (patch pcr-000150722_stu8130201)

Drop Roster functionality can be configured through the Banner Student administrative pages which were released in Banner Student 9.3.6.

If you plan to use the Class List administrator role to access the Class List page, or plan to link to Banner 8.x Self-Service pages from Student Profile, you will also need to apply the following upgrades.

- Banner Student Self-Service 8.7.2
- Banner Faculty and Advisor Self-Service 8.7.2
- Banner Web General 8.6.2

To use some of the enhancements added to the View Grades page in Student Self- Service 9.5, you must enable the **Process GPA by Study Path** check box. This control has been added to the Academic History Control (SHACTRL) page in Banner Student 9.3.3.

Security setup

In order for Self-Service Banner to function properly when PII is enabled, both BANPROXY and BAN_SS_USER must be exempted from PII processing through the setting on the GOAFPUD form.

Single sign on (SSO) support

Banner 9.x applications natively support Single Sign On (SSO) protocols. Currently, Central Authentication Service (CAS) and Security Assertion Markup Language (SAML) 2.0 are the Single Sign On (SSO) protocols supported by Banner Student Self-Service. All integrating applications must be configured to use either of these SSO protocols for authentication services, with the help of a supported centralized Identity Management System.

Note: Banner administrative applications (8.x and 9.x) require CAS for Single Sign On. Banner Student Self-Service has been certified to run on CAS and SAML 2.0 SSO protocols with Ellucian Ethos Identity.

Note: SSO for Banner 8.x forms also requires the SSO Manager, a component of Banner Enterprise Identity Services (BEIS).

Ellucian provides a centralized Identity Management System called Ellucian Ethos Identity that is available for download from the Ellucian Support Center. Ellucian Ethos Identity supports many industry-standard SSO protocols, such as CAS, SAML 2.0, OAuth, OpenID, and so on. Ellucian Ethos Identity provides a single authentication page for end users and Single Sign On (SSO) for applications that recognize the supported protocols. Banner Student Self-Service has been tested and certified with Ellucian Ethos Identity as the primary Identity Management System for SAML 2.0 support.

It is recommended that you first establish the single sign on environment before implementing Banner Student Self-Service. For information on establishing a single sign on environment, refer to the *CAS Single Sign On Handbook* or the *Setting Up Ellucian Ethos Identity* document, available for download from the Ellucian Support Center.

Supported browsers

The following browsers are supported with the application.

- Chrome
- Firefox
- Internet Explorer 10 and 11
- Safari 7 and 8

For more information about supported browsers, refer to the *Interactive Banner Compatibility Guide* on the Ellucian Download Center.

Chrome Java Support

The Java plug-in for web browsers relies on the cross platform plug-in architecture NPAPI, which has been and is currently is supported by all major web browsers.

Google announced plans in September 2013 to remove NPAPI support from Chrome by "the end of 2014", thus effectively dropping support for Silverlight, Java, Facebook Video, and other similar NPAPI based plugins.

As of April 2015, starting with Chrome Version 42, Google has added an additional step when NPAPI based plugins (such as Java) are configured to run. Refer to the article on Oracle Java's website for "Enabling NPAPI support in Chrome".

https://java.com/en/download/faq/chrome.xml

Chrome Java support on Mac OS X

Chrome does not support Java 7 on Mac OS X. Java 7 runs only on 64-bit browsers, and Chrome is a 32-bit browser.

If you download Java 7, you cannot run Java content in Chrome on Mac OS X. You must use a 64-bit browser (such as Safari or Firefox) to run Java content within a browser. Additionally, installing Java 7 disables the ability to use Apple Java 6 on your system.

Chrome Compatibility Mode

When using Chrome, users must disable Compatibility Mode. If Compatibility Mode is not disabled, errors may occur.

To disable Compatibility Mode in Chrome, perform the following steps:

1. Right-click the Chrome shortcut and select **Properties**.
2. Select the **Compatibility** tab.
3. Clear the **Run this program in compatibility mode for:** check box.
4. Click **Apply**.
5. Click **OK**.

Internet Explorer Compatibility View

When using Internet Explorer, you must be in Internet Explorer Standard Mode. If not, you might receive the following message:

You are viewing this webpage in Compatibility View. Please turn off Compatibility View in your browser (Tools menu) for optimal viewing experience.

To disable Compatibility View in Internet Explorer 9, perform the following steps:

1. Select **Tools** > **Compatibility View Settings**.
2. Clear the **Display intranet sites in Compatibility View** and **Display all websites in Compatibility View** check boxes.
3. Click **Close**.

You can also deactivate the **Compatibility View** item in the **Tools** menu.

Java dependencies

Java 1.7.x (64-bit version) must be installed on the application server before you install the application. The same version of Java must be used to customize and deploy the WAR file.

The JDK bin directory must be defined in the PATH system property.

Note: Development for Banner 9.x is currently supported on Java 7 *only*.

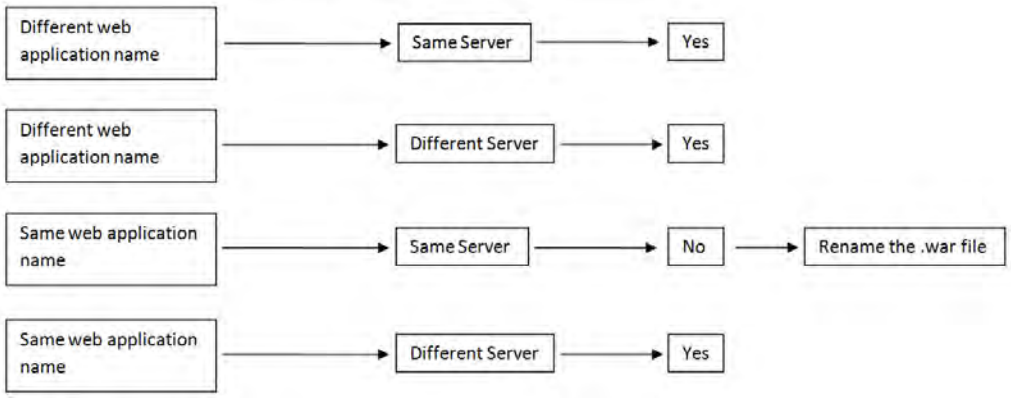
Note: Java 7 includes security restrictions for Rich Internet Applications. Refer to Article 000030656 on the Ellucian Support Center for details on Java 7 security restrictions with Liveconnect calls to Oracle Forms Applet.

Enable Application Navigator

To enable seamless navigation for Banner Student Self-Service using Application Navigator, you will need to add an entry to the Application Navigator Configuration file. You will also need to complete additional configuration steps for Web Tailor menus. For instructions on how to perform these tasks, refer to the *Application Navigator Installation Guide* version 2.0.1 or greater, and the “Integration with Self-Service” section of the *Application Navigator Handbook.*

Deployment of multiple web applications

The following diagram describes various scenarios of deploying multiple web applications:



In the first and second scenarios, you can deploy multiple web applications with different WAR file names on the same or different servers.

In the third scenario, if you want to deploy multiple web applications on the same server, the WAR file names must be different.

In the fourth scenario, you can deploy multiple web applications with the same WAR file name on different servers.

With CAS or SAML2 SSO

If CAS or SAML2 Single Sign On (SSO) is enabled, a user is authenticated only for the first log in. Subsequently, the user can access and navigate among any Banner 9.x applications without logging in again. When SSO is enabled and configured between Banner administrative applications, when a user logs out of one application, the user is logged out of all applications that are open.

Refer to the *CAS Single Sign On Handbook* or the *Setting Up Ellucian Ethos Identity* document, available on the Ellucian Support Center, for details on configuring Banner Applications with Single Sign-on for multiple protocols (SAML 2.0, CAS, etc.).

Without SSO

If Single Sign On (SSO) is not enabled, a user must authenticate before accessing each Banner 9.x application. If SSO is not enabled when a user logs out of an application, the user is logged out of that current application. The user is still logged in to all other applications that are currently open.

F5 load balancer configuration

The application was tested using an F5 load balancer configured with the following settings:

```
Load Balancing type = Round Robin

Persistence = Cookie
```

Note: Other configurations may be supported depending on Network Load Balancing (NLB).

Upgrade the Database

The following steps are used to upgrade the database:

- "Perform the Banner DB Upgrade steps" on page 18
- "Update login.sql" on page 18
- "Verify that the required products are applied" on page 19
- "Verify the ban_ss_user database account" on page 19
- "Set up access for application users with an administrative account" on page 19
- "Migrate staged files to the permanent directories" on page 19
- "Update the version number" on page 21

Perform the Banner DB Upgrade steps

Some database upgrade steps are common to all Banner 9.x applications. These common database upgrade steps must be performed before you upgrade the database for the Banner Student Self-Service application.

Refer to the instructions in the *Banner DB Upgrade, Upgrade Guide* (`Banner_db_upgrade_9.12_Upgrade_Guide.pdf`) for the common database upgrade steps. The *Banner DB Upgrade, Upgrade Guide* is delivered in the `banner-db-upgrade-91200d.trz` file.

Update login.sql

You must edit `login.sql` to update the schema owner's default password and to specify the path to create log files. To update the delivered `login.sql` script, perform the following steps:

1. Replace the #UPDATEME# string with the value of a particular schema owner's password in your environment. Make this update in your environment for each Banner schema owner.
2. Set the value that gets assigned to `splpref`. The value can be set to the ORACLE_SID or to a directory name. Your options depend on the operating system.

   The `splpref` variable defines the file prefix that the installation process uses to generate listings or intermediate SQL routines. This feature allows you to segregate the generated output when the stage must be applied to more than one instance.

Verify that the required products are applied

To check that all prerequisite products are applied to the environment, perform the following steps:

1. Invoke SQL*Plus and run the following procedure:

   ```
   sqlplus /nolog @ruappready
   ```

2. Review the `ruappready` listing.

Verify the ban_ss_user database account

The `ban_ss_user` account is used for database connections for self-service applications. The database upgrade process grants the `BAN_DEFAULT_M` role to `ban_ss_user`. If this role is revoked, the application will not start successfully.

Set up access for application users with an administrative account

A new security object named `SELFSERVICE` is created during the installation of the self-service application. Application users who have an administrative account associated with their login on the Enterprise Access Controls (GOAEACC) page must be assigned this new object with `BAN_DEFAULT_M` privilege.

Note: The `SELFSERVICE` object was also added to the `BAN_GENERAL_C` class. As an alternative, you can associate your administrative users with this class.

Migrate staged files to the permanent directories

This release provides migration scripts for Unix and Windows platforms. These scripts expect your directory structure to match the directory structure created by the Banner installation process. If you choose a different directory structure, you must modify the scripts. The release does not include migration scripts for other platforms due to their highly customized structures. You can, however, use the file `STUMIGR.TXT` as a starting point for writing your own migration scripts.

Unix

The file `STUMIGR.TXT` lists all files that must be deleted from your permanent directories, and all files that should be copied from the staging directory to your permanent directories. The destination is indicated in UNIX format. The format is different on other platforms.

The file `stumigr.shl` does the appropriate removes, copies, and links. The local LN variable at the top of `stumigr.shl` determines the type of links that are used in the migration. If you want to use symbolic links, set `LN='ln -s'` so that the command `${LN} file $BANNER_HOME/links` is translated to `ln -s file $BANNER_HOME/links`. If you want to force the removal of any existing targets before linking files, set `LN='ln -f'`.

To run the migration script in background on a Unix platform, perform the following steps:

1. Ensure that the directory path names in `stumigr.shl` are correct.
2. Ensure that the environment variable `$BANNER_HOME` in `stumigr.shl` is set to the appropriate directory.
3. Sign on to an operating system account that has write permission into the target Banner directories.
4. If you are a cshell user (your operating system prompt is a percent sign), enter `sh` and press Enter to enter the Bourne shell.
5. Navigate to the staging directory for the product.
6. Run the migration script as follows:

   ```
   sh stumigr.shl >stumigr.log 2>&1 &
   ```

7. If you were a cshell user and want to return to that mode, press CTRL-D or enter `exit`. Then press Enter.
8. Review `stumigr.log`. This file contains the results of the migration.

Note: Even if your directory structure matches the baseline perfectly, some link commands will fail (that is, where the link currently exists). Other link errors might indicate that you had two copies of an object when the migration script was executed. This condition must be corrected. The duplication is probably between links and the product subdirectory.

Windows

The file `stumigr.pl` does the appropriate deletes and copies. To run the migration script on a Windows platform, perform the following steps:

1. Check the value of the BANENV environment variable by executing the SET command from the DOS prompt.
   - If the value of BANENV is REG, the value used for BANNER_HOME will be taken from the registry entry:

     HKEY_LOCAL_MACHINE\SOFTWARE\BANNER\BANNER_HOME
   - If the value of BANENV is ENV, the value used for BANNER_HOME will be taken from the environment variable BANNER_HOME.
2. Ensure that the directory path names in `stumigr.pl` are correct.
3. Sign on to an operating system account that has write permission into the target Banner directories.
4. Navigate to the staging directory for the product.
5. Run the migration script as follows:

   ```
   perl stumigr.pl >stumigr.log 2>&1
   ```
6. Review `stumigr.log`. This file contains the results of the migration.

Update the version number

To insert the release version number into the Web Application (GURWAPP) table, perform the following steps:

1. Invoke SQL*Plus and run the following procedure to insert the version number for the Banner Student Self-Service application:

   ```
   sqlplus general/password
   start versionupdate
   ```
2. Review the `versionupdate` listing.

Install Banner Student Self-Service Application for CAS SSO

The following sections detail the installation steps for the Banner Student Self-Service 9.7 application:

- "Undeploy the existing application" on page 22
- "Customize the WAR file" on page 25
- "Setup CAS SSO Configuration" on page 43
- "Regenerate the WAR file" on page 47
- "Configure and deploy the WAR file to a web application server" on page 48
- "Configure the application" on page 63

Undeploy the existing application

Before you install Banner Student Self-Service 9.7, you must undeploy previous 9.x versions of Banner Student Self-Service and Banner Student Attendance Tracking Self-Service (if installed), and Banner Student Advisor Self-Service (if installed). If no previous 9.x versions of Banner Student Self-Service, Student Attendance Tracking Self-Service, or Banner Student Advisor Self-Service are installed, you can skip this section.

The following sections give the required steps to undeploy existing Banner 9.x applications in Tomcat and WebLogic servers.

- "Tomcat" on page 23
- "WebLogic" on page 24

Tomcat

You can either use the Tomcat Manager web application to undeploy the existing application or you can shut down Tomcat and manually remove the files.

Undeploy using the Tomcat Manager web application

Use the following procedure to undeploy the application using the Tomcat Manager web application:

1. Access the Tomcat Manager web application at one of the following URLs:

   ```
   http://server:8080/manager
   ```

   or

   ```
   http://server:8080/manager/html
   ```

2. Access the deployment page using a valid user name and password.
3. Under the Commands area, click **Stop** to stop the existing application.
4. In the confirmation dialog box, click **OK**.
5. Under the Commands area, click **Undeploy**.
6. In the confirmation dialog box, click **OK** to undeploy the application.

Undeploy using a manual procedure

The following sections give the steps to manually undeploy the existing application on Unix and Windows operating systems.

Unix

Use the following procedure to manually undeploy the existing application on a Unix operating system:

1. Log in to the server where Tomcat is running, using the same account credentials that were used to start Tomcat.
2. Shut down Tomcat by running the shutdown script:

   ```
   $CATALINA_HOME/bin/shutdown.sh
   ```

3. Remove the current deployment and associated WAR file:

   ```
   cd $CATALINA_HOME
   rm -rf $CATALINA_HOME/webapps/StudentSelfService
   rm -rf $CATALINA_HOME/webapps/StudentSelfService.war
   ```

4. If you have a previous version Student Attendance Tracking (9.2 or older) installed, remove the deployment and associated WAR file.
5. If you have a previous version Student Advisor (9.3.0.2 or older) installed, remove the deployment and associated WAR file.

Windows

Use the following procedure to manually undeploy the existing application on a Windows operating system:

1. Use the command prompt to shut down Tomcat.

   Note: If you installed Tomcat as a service, use the Service Control panel to stop the application. Otherwise, use the shutdown script `%CATALINA_HOME%\bin\shutdown.bat`.

2. Remove the current deployment and associated WAR file:

   ```
   rmdir %CATALINA_HOME%\webapps\StudentSelfService /s/q
   del %CATALINA_HOME%\webapps\StudentSelfService.war /q
   ```

3. If you have a previous version Student Attendance Tracking (9.2 or older) installed, remove the deployment and associated WAR file.
4. If you have a previous version Student Advisor (9.3.0.2 or older) installed, remove the deployment and associated WAR file.

WebLogic

Use the following procedure to stop and undeploy the Banner 9.x application:

1. Access the administration server using the following URL:

   ```
   http://server:7001/console
   ```

2. In the Domain Structure frame, click **Deployments**.
3. In the Change Center, click **Lock and Edit**.
4. Select the check box to the left of the Banner 9.x application.
5. Click **Stop**.
6. Click **Force Stop Now**.
7. In the Force Stop Application Assistant page, click **Yes**.
8. Select the check box to the left of the Banner 9.x application.
9. Click **Delete**.
10. In the Delete Application Assistant page, click **Yes**.
11. In the Change Center frame, click **Activate Changes**.

Customize the WAR file

The name of the release package is `release-StudentSelfService-9.7.zip`. This release package is moved to the `$BANNER_HOME\student\java` subdirectory during the database upgrade. Use the following steps to unzip the release package and customize the WAR file for your institution.

Note: JDK 1.7 must be installed on your system. See the Java dependencies section for more information.

Unzip the release package

To unzip the release package into a temporary directory, perform the following steps:

1. Log in to the application server platform.

   Note: You must have a valid application server account to deploy into the application server container (Tomcat or WebLogic).

2. Create a temporary directory. For example:

   ```
   mkdir $HOME/ban9temp
   ```

3. Locate the release package `release-StudentSelfService-9.7.zip.`
4. Transfer this file in binary mode using File Transfer Protocol (FTP) file into the temporary directory. For example:

   ```
   $HOME/ban9temp
   ```

5. Unzip `release-StudentSelfService-9.7.zip` into the temporary directory.

Prepare the installer

To prepare the installer, perform the following steps:

1. Change the directory to the installer directory:

   ```
   cd installer
   ```

2. Run the `ant` command, which will build the installation tool.

   Note: For Unix, make sure the ant file is executable. For example, `chmod +x ant`.

Example:

```
ban9temp $ cd installer
ban9temp/installer $ ./ant
```

The message *Build successful* confirms a successful build.

Install into the product home directory

The product home directory supports the configuration and creation of a deployable WAR file. Although Banner 9.x web applications are modular and are installed independently, they share a common configuration. The package provides a common installer that creates consistent product home directory structures for all Banner 9.x applications.

Within a particular environment, you should place the product home directories for Banner 9.x applications in sibling directories. For example, the following directory structure includes four product home directories and a `shared_configuration` directory that support a common test environment.

```
TEST/
|--> Catalog/
|--> Schedule/
|--> StudentOverall/
|--> StudentRegistration/
|--> shared_configuration/
```

A product home directory is created for each deployment. For example, the home directory that is used for the application within a test environment is different than the home directory that is used for the production environment. When you are supporting different environments for multiple home directories for the same solution, this structure provides the necessary configuration, release level, and custom modification flexibility.

The following directory tree illustrates the product home directory that is created for the test environment:

```
TEST/                                   (optional and recommended top-level directory for all homes)

|-->app-name                            (product home for 'app-name' in test environment)
    |--> current
        |--> instance/                  (instance-specific configuration that will not be overwritten)
            |--> config/
                |--> {app-name}_configuration.groovy (module-specific configuration for CAS, logging, etc.)
            |--> i18n                   (new or replacement message bundles that should be added the war)
            |--> css                    (new or replacement css files that should be added the war)
            |--> js                     (new or replacement javascript files that should be added the war)

        |--> lib
            |--> ojdbc6.jar           (the Oracle database driver that must be placed manually into the tomcat/lib directory)
            |--> logging.properties   (logging configuration that may be copied to the WEB-INF/classes directory that is
                                       very useful if the war file cannot be deployed successfully.)
        |--> i18n/                      (contains message bundles that may reflect changes not yet in 'baseline')
        |--> dist/                      (contains the war file, after it is creating using the 'systool')
        |--> installer/                 (contains the installer)
    |--> archived-releases/             (directory for previous releases)
        |-->

|--> shared_configuration/              (home for configuration files shared across modules within an environment)
    |--> banner_configuration.groovy    (a 'shared  configuration file containing dataSource)
```

In addition to the application's product home directory, a separate `shared_configuration` home directory contains cross-application configuration for

the test environment. This directory holds the `banner_configuration.groovy` file, which contains the shared JNDI datasource configuration.

To install the installer into the product home directory, perform the following steps:

1. Ensure that the installer is prepared using `ant`.
2. Use the installer to install the release file into the product home directory.

   Note: Your current working directory must be in the installer directory (`ban9temp/installer`) before executing the following commands.

   On Unix:
   ```
   $ bin/install home
   ```

   On Windows:
   ```
   > bin\install home
   ```

3. When prompted, enter the full path of the application home directory. The application will be installed within the `current` subdirectory within this home directory and the previous release will be archived.

   On Unix:
   ```
   []: Current_home_directory/banner_test_homes/
   StudentSelfService
   ```

   On Windows:
   ```
   []: c:\banner_test_homes\StudentSelfService
   ```

4. Enter the full path of the `shared_configuration` home directory. Banner 9.x applications that refer to this home directory share this configuration file.

   On Unix:
   ```
   []: Current_home_directory/banner_test_homes/
   StudentSelfService
   ```

   On Windows:
   ```
   []: c:\banner_test_homes\StudentSelfService
   ```

   Note: If an identified home directory or the `shared_configuration` home directory does not exist, the installer creates it. The name of a product home directory is not restricted. You can name it when prompted by the installer.

Configure shared settings

The `shared_configuration` home directory contains a cross-application configuration file called `banner_configuration.groovy`. You can change settings in this file.

JNDI datasource

You can optionally change the datasource name in the configuration file to point to the JNDI datasource that is configured in your application server. For example, `jndiName = "jdbc/bannerSsbDataSource"` is the default configuration. You can change this to match the JNDI datasource name in your environment. If you change the jndiName, you must regenerate the WAR file, even if you are using an external configuration.

Link to Self-Service Banner 8.x

To display existing Self-Service Banner 8.x menus and breadcrumbs in Banner 9.x self-service applications, the following configuration must be updated with the URL to your existing Self-Service Banner 8.x application:

```
//replace with the URL pointing to a Self-Service Banner 8.x instance
banner8.SS.url = '<scheme>://<server hosting Self-Service Banner 8.x>:<port>/<context root>/'
```

For example:

```
banner8.SS.url = 'http://localhost:8002/ssb8x/'
```

Note: If the `banner8.SS.url` setting is not in the `banner_configuration.groovy` file, you must add it to the file before you continue with the installation.

To provide single sign on (SSO) to Banner 8.x, Jasig Central Authentication Service (CAS) (www.apereo.org/cas) is required as an external authentication provider. The following components must be configured to authenticate using CAS:

- Student Self-Service application must be configured to authenticate using CAS.
- The SSO Manager must be deployed and configured to enable Self-Service Banner 8.x authentication using CAS. The SSO Manager is a component of Banner Enterprise Identity Services (BEIS).

```
banner8.SS.url = 'http://beissmpl.university.com:7777/ssomanager/c/SSB? pkg='
```

The `banner8.SS.url` setting references the SSO Manager URL (`'http://beissmpl.university.com:7777/ssomanager/c/SSB'`) and appends the `'?pkg='` suffix to support deep linking to a specific Self-Service Banner 8.x page.

The following is an example of a Banner 8.x SSO URL through the SSO Manager:

```
http://beissmpl.greatvalleyu.com:7777/
ssomanager/c/SSB? pgk=bwgkogad.P_SelectAtypView
```

Banner Self-Service now supports locale specific redirect to Banner 8x from Student Self-Service. The following is an example of a Banner 8 SS Locale URL:

```
banner8.SS.locale.url =[
default : 'http://localhost:8002/
DEFAULT/', en : 'http://
localhost:8002/EN/',
en_US        :        'http://
localhost:8002/enUS/', en_AU :
'http://localhost:8002/enAU/',
en_GB        :        'http://
localhost:8002/enGB/', en_IE :
'http://localhost:8002/enIE/',
en_IN        :        'http://
localhost:8002/enIN/',   fr   :
'http://localhost:8002/FR/',
fr_CA : 'http://localhost:8002/
frCA/', pt : 'http://
localhost:8002/PT/',
es : 'http://
localhost:8002/ES/', ar :
'http://localhost:8002/
AR/',
]
```

Note the following:

Following are important notes about the redirect from Student Self-Service to Banner 8.x.

- If a locale specific URL does not exist, then it will use the default URL defined in the `banner8.SS.locale.url` map.
- For SPRIDEN users, if you do not define the `banner8.SS.url` setting with a valid URL, Banner 8.x menus and breadcrumbs are not displayed in the Student Self-Service application.
- Without CAS and the SSO Manager, navigation to Banner 8.x is not seamless. A user must log in to Banner Self-Service 8.x using a valid user name and password. Navigation terfminates at the Banner Self-Service 8.x main menu page.

Navigational MEP support from Student Self-Service to Banner Self-Service 8.x

Self-Service application supports the ability to call a MEP context-sensitive URL that maps to the appropriate Banner Self-Service 8.x URL. The URL configurations enable the end user SSO access from Self-Service to Banner Self-Service 8.x applications along with preserving the MEP context for that user session.

If your institution uses MEP, you must update key configurations and URL contexts. The key for the string must be in the following format: `mep.banner8.SS.url` and the key must be configured with the following map

list: `mep.banner8.SS.url = ['GVU': 'http://<host_name>:<port_number>/<banner8>/SMPL_GVU', 'BANNER': 'http://<host_name>:<port_number>/<banner8>/SMPL_BANNER' ]`

The following is an example of a Banner 8.x SSO URL through the SSO Manager in a MEP environment: `mep.banner8.SS.url = ['GVU': 'http://e009070.ellucian.com:8888/ssomanager/c/SSB?pkg=http://e009070.ellucian.com:9020/SMPL_GVU/', 'BANNER': 'http://e009070.ellucian.com:8888/ssomanager/c/SSB?pkg=http://e009070.ellucian.com:9020/SMPL_BANNER/' ]`.

Use the following structure to support a locale specific redirect to Banner 8:

```
mep.banner8.SS.locale.url=[ GVU:
                [
                     “default” : 'sampleUrlDefault ', "ar":
                     'sampleUrlAR ',
                     "fr": 'sampleUrlfr', "fr_CA":
                     'sampleUrlfrca'
                ],
      BANNER :
                [
                      “default” : 'sampleUrlDefault', "ar":
                      'sampleUrlAR',
                      "fr": 'sampleUrlfr ', "fr_CA":
                      'sampleUrlfrca'
                ]

    ]
```

Note: If a locale specific URL does not exist, then it will use the default URL defined in the `mep.banner8.SS.locale.url` map.

Session timeouts

The following timeouts are used in the self-service application:

- “banner.transactionTimeout” on page 31
- “AJAX timeout” on page 31
- “defaultWebSessionTimeout” on page 31

When you determine the timeout settings for CAS and Banner 9.x applications, consider the time limits imposed for each. For example, if the Banner 9.x timeout setting is less than the CAS timeout setting, SSO authentication might still be valid even though the user is exited from the Banner 9.x application. As long as the CAS authentication remains valid, the user can re-enter the Banner 9.x application without re-authenticating.

banner.transactionTimeout

The `banner.transactionTimeout` setting is used to prevent excessive delays due to long database transactions. The setting is configured in the `banner_configuration.groovy` file. To use this timeout, set the `banner.transactionTimeout` setting to `300` seconds.

Ensure that either the configuration file is deployed with the application, or the application is using the configuration file where it is currently located.

If a database transaction takes longer than `banner.transactionTimeout` seconds, the transaction is aborted and any change is rolled back.

Note: In some cases, the web user interface might ignore the database timeout/error notification and not remove the loading spinner. If this occurs, refresh the page to continue using the application.

AJAX timeout

The AJAX timeout terminates HTTP requests that exceed the specified time limit. The self-service application sets the timeout value based on the `banner.transactionTimeout` setting plus an increment to allow for communication and processing of the request.

You do not need to override the AJAX timeout, but the timeout value can be overridden in JavaScript by calling `$.ajaxSetup( { timeout: timeoutValue } );`

Note: The `timeoutValue` must be in milliseconds.

Note: Although the web user interface continues after an AJAX timeout, the server might continue processing the request until it completes or reaches a database `transactionTimeout`.

defaultWebSessionTimeout

The `defaultWebSessionTimeout` property is used to terminate user sessions that are left idle or abandoned without logging out. This setting is used to set the overall session inactivity time. The `defaultWebSessionTimeout` setting can be configured in the `banner_configuration.groovy file`.

Note: If `defaultWebSessionTimeout` is not specified, a default value of *1500* seconds is used.

Role-based web session timeouts are configurable in Banner Web Tailor where `TWTVROLE` is the logged in user role and `TWTVROLE_TIME_OUT` is the timeout for users with that role.

Note: An individual's session timeout is the longer of the `defaultWebSessionTimeout` and the role-based timeout from `TWTVROLE`.

Number of supported email recipients

The `email.batch`.size setting specifies the maximum number of recipients that can be supported by your institution's email system. You must add this setting to the `banner_configuration.groovy` file.

Configure footerFadeAwayTime

To configure the footer fade away time. This will be defaulted to 2 seconds.

The default value is: footerFadeAwayTime = 2000

Application's Secured Landing page:

1. If footerFadeAwayTime is configured then user will be able to view the footer when accessing the application that is secured page, once in 24hrs
2. If footerFadeAwayTime is not configured the user will be able to view the footer for 2 seconds when accessing the application that is secured page, once in 24hrs
3. If within 24hs the cache is cleared and the secured page of the application is accessed by the same user then the user can view the footer for the configured time
4. If the user uses a different machine but the same credential then the user will view the footer for the configured time

Application's Unsecured Landing page:

User will be able to see the footer for the configured time or default (time 2 seconds) on the applications unsecured landing pages on every access

Location of photographs

The `banner.picturesPath` setting specifies the location of student, faculty member, and advisor photographs. For example:

```
banner {
   picturesPath = System.getProperty('base.dir') + '/test/images'
}
```

In this example, the `banner.picturesPath` setting equals the application base directory appended with `/test/images`.

Default photograph

The `banner.defaultPhoto` setting specifies the default photograph that is displayed if a student, faculty member, or advisor photograph does not exist.

Application properties

The `bannerStudentAdvisorUI_configuration.example` file contains properties that can be configured for the Banner Student Advising Student Profile and Banner Student Self-Service applications. This file must be renamed to `bannerStudentAdvisorUI_configuration.properties` and moved to the directory where the `banner_configuration.groovy` file is located.

Execute the following statement from the `$HOME/ban9temp/config` directory:

```
cp bannerStudentAdvisorUI_configuration.example
[shared_configuration]/
bannerStudentAdvisorUI_configuration.properties
```

where `[shared_configuration]` is the directory location of the `banner_configuration.groovy` file.

Once the file has been renamed and moved, you can modify the properties for your environment.

Note: It is common to share the `bannerStudentAdvisorUI_configuration.properties` file with the Student Advisor application.

Configure application-specific settings

The installer creates the `StudentSelfService_configuration.example` file. Copy this file and change the name to `StudentSelfService_configuration.groovy`. Place the copied file in the `StudentSelfService\current\instance\config` directory. This application-specific configuration file contains settings that you can customize for your specific environment.

Cross-frame scripting vulnerability

For the Student Self-Service application to appear inside the Application Navigator, institutions need to override the default settings to set the following configuration in the `StudentSelfService_configuration.groovy file`.

```
grails.plugin.xframeoptions.urlPattern = '/login/auth'

grails.plugin.xframeoptions.deny = true
```

Note: This value must not change.

Configure Extensibility in Student Self-Service application

Following configuration change is needed before using the tool. Each application can have the JSON file directory specified in its own [application]_configuration.groovy file or a default directory can be specified in the banner_configuration.groovy file. For example, in either the application-specific configuration file or the banner_configuration.groovy, include the following:

Example external configuration files

```
webAppExtensibility {
    locations {
       extensions = "C:/BanXE/Extensions/ss_ext/extensions/"
        resources = "C:/BanXE/Extensions/ss_ext/i18n/"
    }
}
```

The "extensions" configuration specifies the directory that will contain the JSON files containing modifications to the application pages. The "resources" configuration specifies the directory that will contain property files for any multilingual text values required for the changes. For example, modifications to field labels and title text.

Security has been configured for all non-production environments to have the WEBTAILORADMIN role as the default role to access the Extension Editor. This role can be changed by an institution if it is required. For the Production environment, no default role is set and no access will be given to any user as a default of the installation. Administrative functions of extensibility, like the ability to edit the extension in the web interface are disabled in the application (when running the application in development or test mode users with the WTAILORADMIN role have access to the administrative functionality to support unit testing), unless explicitly allowed in the webAppExtensibility adminRoles configuration:

Example external configuration files

```
webAppExtensibility {
    …
    //Comma separated list of roles
adminRoles = "ROLE_SELFSERVICE-WTAILORADMIN_BAN_DEFAULT_M"
}
```

Bookstore links

The Higher Education Opportunity Act (HEOA) requires your institution to configure at least one hyperlink to an online bookstore. You can link to a bookstore web site in several ways:

- With no parameters
- With hard-coded parameters
- With dynamic parameters
- For a specific campus code

Bookstore links are configured in the `StudentSelfService_configuration.groovy` file. Links configured in this file are available at the section level for a course.

Use the following parameters to set up a bookstore link:

Parameter	Required/ optional	Instructions
URL	Required	Enter a valid URL for the link to the bookstore.
Label	Required	Enter the `message.properties` code to be displayed as the hyperlink label. Add the code and the label text to the `messages.properties` file, using the `bookstore.links.follett` sample as a guide.
Campus	Optional	Enter specific links based on the campus code. For example, if you have multiple campuses and you wish to use a different Follett bookstore number for each campus, include a URL configuration for each campus and include the campus code in the URL parameters.
Page	Required, if the Params parameter is used	Enter the Overall Page and Field Configuration (SOAWSCR) page number, if the URL is specific to the search results (`30` equals Section).

Parameter	Required/ optional	Instructions
Params	Optional	Three options are available for identifying the parameters to be entered in the URL: • Select a value from SOAWSCR for substitution. (`INSTRUCTOR` and `MEETINGTIME` are not valid values.) • Identify the element directly from the section object code, if it is not on SOAWSCR. • Supply NAME in the parameter list, to pass student name to the bookstore link. The format of the name is specified in `default.name.format` in the General Person plugin `messages.properties` file. If using variables, use {0}, {1}, and so on, as placeholders for the parameters, appearing in the same order as in the params array.

Theme Editor tool

Use the Theme Editor tool to apply color theme and logo changes to Banner applications.

You can use the following settings to enable the Theme Editor.

Setting	Description
banner.theme.url="<UPDATEME>"	The URL of the application hosting the Theme Server.
banner.theme.name="<UPDATEME>"	The name for the theme. In a Multi-Entity Processing (MEP) environment, the application uses the MEP code as the theme name instead of banner.theme.name. You must create a theme with the same name in the Theme Editor on the server specified by banner.theme.url.
banner.theme.template="StudentSelfService	The scss file containing the theme settings in the WAR file.
banner.theme.cacheTimeOut=120s	Indicates how long to cache the CSS file that was generated using the template and theme.This is an advanced setting that you can use when the application is configured to be the host of the theme server.s

After you enable the Theme Editor, the Preview section displays an example of how the theme settings might look. Your theme choices could display differently in some

applications. You can configure the following theme parameters in the Theme Editor section.

Parameter	Description
Theme name (required)	The name is the theme's primary identifier. Create a recognizable name for the theme, for example, an institution name or abbreviation. If your institution is MEP-enabled, you may want to use a MEP code as a theme name. This gives you the ability to create code themes based on MEP code. If you change the name of an existing theme and click Save, this creates a new theme.
Primary color	Defines the color of the main title bar. Use the drop down or enter the hex code of the color. Theme Editor generates several related shades for borders, backgrounds, and text based on your color choice. View the Preview section to see your choices. It is recommended that you choose colors that coordinate with your institution logo.
Secondary color	Defines color of other page elements related to the Primary color.
Accent color	Defines additional color for various page elements. As you modify the primary and secondary colors, the Theme Editor automatically determines an accent color that contrasts with the primary and secondary colors. If you prefer, you can specify the Accent color.
Logo URL	The URL of the to include in pages. Create the logo as an SVG file so that it can scale to fit the page The URL must be accessible to application clients because users' browsers will load the logo from this URL.
Save Theme	Saves the theme based on the current Theme name. The theme is saved in the file system of the application server hosting the Theme Editor and is available to all applications. When you click **Save Theme**, the theme is applied to the current page.
New Theme	Clears the theme settings. You can also create a new theme by copying an existing one, changing the Theme Name, and clicking **Save Theme**.

Parameter	Description
As you create themes, they display in the **Saved Themes** section where you can perform the following functions.	
Load	Loads the theme into the Theme Editor and applies the theme to the current page.v
Delete	Deletes the theme from Theme Server. This action cannot be undone.
JSON	Links to the JSON values that encode the theme. This allows you to save the JSON to another location.
CSS	Links to the CSS file generated for the theme. This lets you save the CSS to another location and optimize performance by configuring applications to use the static theme. However, if an application uses a statically saved theme file, you must manually update the static file with any theme changes.

Google Analytics

Use Google Analytics for the Student SelfService application.

Access the `StudentSelfService_configuration.groovy` file and use the following configuration to enable Google Analytics for the StudentSelfService application.

Name	Description
`banner.analytics.trackerId=`	Set this value to your institution's Google Analytics Tracker ID. Default is blank
`banner.analytics.allowEllucianTracker=`	Specify whether to allow Ellucian Tracker. Default is true

Configuration example

```
banner.analytics.trackerId = ‘UA-83915850-1’
banner.analytics.allowEllucianTracker = false
```

Following are the possible combination of configuration values and their outcome.

- If there is no configuration in the configuration file, then by default the Ellucian tracking ID is enabled and Ellucian tracks analytics.
- If `allowElluciafnTracker=true`, Ellucian tracks the analytics data.

- If `allowEllucianTracker=false`, the tracking script will not be added in the gsp page.
- If only the client tracker ID exists in the configuration, then both Ellucian and the client track the Google Analytics data.
- If `allowEllucianTracker=true` and the client tracker ID exists in the configuration, then both the client and Ellucian track the analytics data.

If `allowEllucianTracker=false` and the client tracker ID both exist in the configuration, then the client tracks the analytics data, but Ellucian does not.

Quartz Scheduler Configuration Settings

Update the `StudentSelfService_configuration.groovy` file for the newly added properties: `configJob.interval`, `configJob.delay`, `configJob.actualCount`.

Name	Description
`configJob.delay`	Defines when the quartz scheduler should start after the server startup. Enter a value of time in milliseconds. If not configured, the default value is 60000.
`configJob.interval`	Defines the interval at which the quartz scheduler should run. Enter a value of time in milliseconds. If not configured, the default value is 60000.
`configJob.actualCount`	Defines the how many times the config job will run. • -1, the job to run indefinitely • 0, the job will not run If not configured, the default value is -1.

Self-service end point

The `ssbEnabled` setting is set to `true` for instances that expose self-service end points. Banner Student Self-Service is a self-service application, so the default value is always `true`.

Integration with Ellucian Degree Works

If you are using Ellucian Degree Works, add the following setting to the `StudentSelfService_configuration.groovy` file:

```
bannerXE.url.mapper.degreeWorksUrl='<fully qualified path to
Ellucian Degree Works>'
```

For example:

```
bannerXE.url.mapper.degreeWorksUrl='<protocol>://
<host>:<port>/dev/dwadvss/banmain/
IRISLink.cgi?CAS=ENABLED&SERVICE=LOGON
&SCRIPT=SD2WORKS&PORTALSTUID=<STUDENTID>'
```

Navigation to other applications

To navigate from Class List application to other applications, the `StudentSelfService_configuration.groovy` file needs to configured as follows:

```
// ********************************************************

//                      +++ Navigation to other apps +++

// ********************************************************
```

The following are keys that must be configured:

```
classListApp.fgeURL = 'http://host:port/StudentFacultyGradeEntry/'
classListApp.attrURL = ' http://host:port/
FacultyAttendanceTrackingSSB/'
```

Configure View Grades page via SQL script

The View Grades menu will be available as part of the DB upgrade process. However, the user must configure the URL via Web Tailor or in the Web Tailor Repeating Menu Item (TWGRMENU) table.

Use the following script to update the URL for the View Grades page.

```
UPDATE "WTAILOR"."TWGRMENU" SET TWGRMENU_URL = 'http://
application_path_of_StudentSelfService/ssb/studentGrades'
WHERE twgrmenu_name = 'bmenu.P_AdminMnu' and
twgrmenu_url_text = 'View Grades';
```

```
Where application_path_of_StudentSelfService =
StudentSelfService application path. (example: http://
host_name:port_number/StudentSelfService)
```

Configure Class List page via SQL script

The Class List menu will be available as part of the DB upgrade process. However, the user must configure the URL via Web Tailor or in the Web Tailor Repeating Menu Item (TWGRMENU) table.

Use the following script to update the URL for the Class List page.

```
UPDATE "WTAILOR"."TWGRMENU" SET TWGRMENU_URL = 'http://
application_path_of_StudentSelfService/ssb/classListApp/
classListPage' WHERE twgrmenu_name = 'bmenu.P_FacMainMnu'and
twgrmenu_url_text = 'Class List';

Where application_path_of_StudentSelfService =
StudentSelfService application path. (example: http://
host_name:port_number/StudentSelfService)
```

Note: From the Class List page, users can navigate to the Attendance Tracking page.

Configure Drop Roster page via SQL script

The Drop Roster menu will be available as part of the DB upgrade process. However, the user must configure the URL via Web Tailor or in the Web Tailor Repeating Menu Item (TWGRMENU) table.

Use the following script to update the URL for the Drop Roster page.

```
UPDATE "WTAILOR"."TWGRMENU" SET TWGRMENU_URL = 'http://
application_path_of_StudentSelfService/ssb/dropRoster/
dropRosterPage' WHERE twgrmenu_name =
'bmenu.P_FacMainMnu'and twgrmenu_url_text = 'Drop Roster';

Where application_path_of_StudentSelfService =
StudentSelfService application path. (example: http://
host_name:port_number/StudentSelfService)
```

JMX MBean name

The name that is used to register MBeans must be unique for each application that is deployed into the JVM. This configuration should be updated for each instance of each application to ensure uniqueness.

```
jmx {
    exported {
        log4j = "student-self-service-log4j"
    } }
```

Location of the logging file

Log4j is the common logging framework used with applications that run on the Java Virtual Machine. For more information, refer to the log4j documentation.

The configuration file includes documentation on various elements that can be modified depending on your environment.

The following is an example of how to override the location where the log file is saved.

```
loggingFileDir = "System.properties['logFileDir'] ?
"${System.properties['logFileDir']}" : "target/logs"
logAppName = "StudentSelfService"
loggingFileName = "${loggingFileDir}/${logAppName}.log".toString()
```

The following is an example of how to override the log file directory properties:

```
export JAVA_OPTS = "-DlogFileDir=/PRODUCT_HOME /"
```

The output logging file location is relative to the application server to which you are deploying.

Logging level

The root logging level is pre-configured to the ERROR level. Multiple class or package level configurations, by default, are set to a status of "off." You can set a different logging level for any package or class. However, the running application must be restarted.

For example:

```
case 'production':
root {
error 'appLog' //change the log level here with the
appropriate log level value.
additivity = true
}
```

Note: Changing the logging level to DEBUG or INFO produces very large log files.

Changes to the `StudentSelfService_configuration.groovy` file require a restart of the application before those changes take effect.

Alternatively, you can use JMX to modify logging levels for any specified package or class, or even at the root level. When using JMX, the logging level changes only affect the running application. When you restart the application, changes that you made using JMX are lost.

For more information on JMX configuration, see “Configure Java Management Extensions” on page 52.

Proxied Oracle users

When connecting to self-service applications, the `ssbOracleUsersProxied` setting specifies whether the Oracle account is used for the connection. This also controls whether Value Based Security (VBS) is enabled in the self-service application. The following values can be used for the `ssbOracleUsersProxied` setting:

- `False` - The Oracle account is not used for the connection and VBS is not enabled in the self-service application. If the Oracle account is locked, the user can still log in to the self-service application.
- `True` - The Oracle account is used for the connection and VBS is enabled in the self-service application. If the Oracle account is locked, the user cannot log in to the self-service application.

Setup CAS SSO Configuration

This section discusses the setting up of CAS configuration details.

Authentication Provider Name

The name identifying the application authentication mechanism. Values are **cas** or **saml**. Specify saml to indicate the application will use SAML 2.0 SSO protocol authentication as shown in the following example:

```
/*****************************************************

*     BANNER AUTHENTICATION PROVIDER CONFIGURATION    *

*                                                     *

*****************************************************/


banner {

sso {

authenticationProvider = 'saml' //  Valid values are: 'saml' and 'cas' for SSO to
work. ’default’ to be used only for zip file creation.

authenticationAssertionAttribute = 'UDC_IDENTIFIER'

if(authenticationProvider != 'default') {

grails.plugin.springsecurity.failureHandler.defaultFailureUrl = '/login/error'

}

if(authenticationProvider ==’saml’ {
grails.plugin.springsecurity.auth.loginForUrl = ’/saml/login’

}

}
```

CAS SSO Configuration

Shown below is a sample of the configuration you can enable for SSO between Student Self-Service and an Identity Management System that supports CAS SSO protocol. Make sure to uncomment this section when CAS SSO is enabled.

CAS SSO Configuration

Shown below is a sample of the configuration you can enable for SSO between Student Self-Service and an Identity Management System that supports CAS SSO protocol. Make sure to uncomment this section when CAS SSO is enabled.

```
/** ******************************************************
*
* CAS CONFIGURATION *
* ****************************************************** **/
// set active = true when authentication provider section configured for cas
grails {
    plugin {
        springsecurity {
            cas {
            active=false
                serverUrlPrefix  = 'http://CAS_HOST:PORT/cas'
                serviceUrl       = 'http://BANNER9_HOST:PORT/APP_NAME/
j_spring_cas_security_check'
                serverName       = 'http://BANNER9_HOST:PORT'
                proxyCallbackUrl = 'http://BANNER9_HOST:PORT/APP_NAME/secure/
receptor'
                loginUri         = '/login'
                sendRenew        = false
                proxyReceptorUrl = '/secure/receptor'
                useSingleSignout = true
                key = 'grails-spring-security-cas'
                artifactParameter = 'SAMLart'
                serviceParameter = 'TARGET'
                serverUrlEncoding = 'UTF-8'
                filterProcessesUrl = '/j_spring_cas_security_check'
                if ((banner.sso.authenticationProvider == 'cas') &&
useSingleSignout){grails.plugin.springsecurity.useSessionFixationPrevention =
false
                }
            }
            logout {
                afterLogoutUrl = 'https://cas-server/logout?url=http://myportal/
main_page.html'
            }
        }
    }
}
```

Logout URL

You can specify where a user should be directed after logging out of the application by updating the `StudentSelfService_configuration.groovy` file. There are two ways the application can handle logouts:

- Logouts can display the CAS logout page with a redirect URL.
- Logouts can automatically go to a redirect URL (without displaying the CAS logout page).

The redirect URL can be different for each Banner application, depending on where you wish to send the user. If the redirect URL is the same for all Banner applications, it can be defined in the global `banner_configuration.groovy` file.

To display the CAS logout page with a redirect URL

With this method of handling logouts, users see the CAS logout page when they log out of the application. The CAS logout page displays a URL that users must click to continue.

Use the `logout.afterLogoutUrl` setting to configure the logout URL. This setting is defined as follows:

```
logout.afterLogoutUrl='https://<CAS_HOST>:<PORT>/<cas>/
logout?url=http://myportal/main_page.html'
```

The logout URL in the following example instructs the CAS server to redirect the browser to `http://myportal/main_page.html` after performing a CAS single logout.

Note: Depending on your needs, you can customize the `serverUrlPrefix`, `serviceUrl`, and `serverName` entries.

Note: It is recommended that you note the values of the artifactParameter, serviceParameter and serverUrlEncoding entries and ensure that they match the values above.

To go directly to a redirect URL

With this method of handling logouts, users automatically go to a redirect URL. Configure logout URL information as follows:

1. Configure logout information, replacing "url" with "service":

   ```
   grails.plugin.springsecurity.logout.afterLogoutUrl = 'https://
   <CAS_HOST>:<PORT>/cas/logout?service=http://myportal/main_page.html'
   ```

2. Set the property `followServiceRedirects` to `true` on the LogoutController that is defined in `cas-servlet.xml`:

   ```
   <bean id="logoutController" class="org.jasig.cas.web.LogoutController"
       p:centralAuthenticationService-ref="centralAuthenticationService"
       p:logoutView="casLogoutView"
       p:warnCookieGenerator-ref="warnCookieGenerator"
   ```

```
    p:ticketGrantingTicketCookieGenerator-
ref="ticketGrantingTicketCookieGenerator"
    p:followServiceRedirects="true" />
```

Password reset

You can specify whether users have the ability to reset their passwords by updating the `ssbPassword.reset.enabled` setting. If the value of the setting is `true`, users can reset their passwords. If the value of the setting is `false`, a "disabled" error message is displayed.

Redirect pages in a MEP environment

Two settings in the configuration file must be configured if your environment is enabled for Multi-Entity Processing (MEP). You do not need to configure these settings if your environment is not enabled for MEP.

If a user tries to access a self-service URL that does not include a valid MEP code, an error prompt is displayed. The user responds by clicking the **Logout** or **Return Home** button. Configuration settings determine where each button redirects the user:

- `grails.plugin.springsecurity.logout.afterLogoutUrl` - This setting contains the URL of the institution-specific page where the user is redirected if the **Logout** button is clicked. An example is a portal page:

  `"https://cas-server/logout?url=http://myportal/main_page.html"` (CAS environment)

  `"http://myportal/main_page.html"` (non-CAS environment)

- `grails.plugin.springsecurity.logout.mepErrorLogoutUrl` - This setting contains the URL of the institution-specific page where the user is redirected if the **Return Home** button is clicked.

Each URL can be any page that does not require a MEP-enabled database connection, or any page outside the self-service application.

Institutional Home Page Redirection Support

With previous versions of the Application, users did not have the option to go back to a home page if they encountered any SSO access issues. This configuration allows Banner Student Self-Service to provide an institutional home page where users can navigate back to, if they encounter access issues specific to insufficient privileges when accessing the application.

```
/*************************************************************
* Home Page URL configuration for CAS / SAML Single-Sign On *
*************************************************************/
grails.plugin.springsecurity.homePageUrl='http://STUDENT_SSB_HOST:PORT/
StudentSelfService' // can be institutional home page ex: http://myportal/
main_page.html
```

Regenerate the WAR file

Once the shared and application-specific configurations are complete, the application WAR file can be regenerated to include your customizations and application-specific settings. The WAR file can then be deployed into your specific application server.

The systool is used to create the WAR file. To set up the systool and to create the WAR file, perform the following steps:

1. Change your current working directory to the product home directory:

   `StudentSelfService/current/installer`

2. Run the ant command, which will build the systool module.

   Note: For Unix, make sure the ant file is executable. For example, `chmod +x ant`.

   Example:

   ```
   $ cd StudentSelfService/current/installer
   StudentSelfService/current/installer $ ./ant
   ```

3. Use the systool module to create the WAR file.

   Your current working directory must be in the `StudentSelfService/current/installer` directory before you execute the following command.

   On Unix:
   ```
   $ bin/systool war
   ```

   On Windows:
   ```
   > bin\systool war
   ```

The WAR file is created in the `StudentSelfService/current/dist` directory.

You can use external configuration files by setting appropriate system properties, although the configuration files are included in the WAR file, making the WAR file self-sufficient. For information on external configuration, see "Configure the Tomcat server" on page 48 or "Create a WebLogic server" on page 56.

Note: If you are using an application that links to the Student application (such as the Student Profile page), there is a different default root context. You will need to change the configuration of those applications to reflect this change. If you will be doing new installations of applications that link to the Student application, please note the new URL context which might not be reflected in the documentation for those applications.

Configure and deploy the WAR file to a web application server

The following sections provides information on configuring the web application and deploying the WAR file to a web application server:

- "Tomcat" on page 48
- "WebLogic" on page 55

Note: Ellucian recommends securing web application traffic using standard TLS encryption supported by the application server software. Please review your application server documentation to learn how to enable HTTPS support for web applications.

Tomcat

The following sections provide information on configuring the web application and deploying the WAR file to the Tomcat server.

Note: If you choose to install the application on a Tomcat server, you do not need to install it on WebLogic.

Configure the Tomcat server

Use the following steps to configure the Tomcat server.

Note: For the Tomcat 8 server configuration, Student Self-Service 9.7 deployment requires specific configurations on unix. Include the following setting,
If ulimit -n < number of files > is less than or equal to 1024, then it should be increased to a value greater than or equal to 4096.

1. Locate the Oracle JDBC jar files (`ojdbc6.jar and xdb6.jar`) in the `StudentSelfService\current\lib` directory.

   Note: Later in the Tomcat configuration process, you will copy the Oracle JDBC jar file into the `\lib` folder under the Tomcat installation directory.

   The account that runs the Tomcat application server must configure environment settings to support the application.

2. On Linux, ensure `CATALINA_HOME` is defined to reference your Tomcat software installation location. For example, `CATALINA_HOME=/opt/apache-tomcat-7.0.xx` where xx indicates the point version of Tomcat you installed.

Warning! *Do not perform this step on the Windows platform.*

Define `CATALINA_OPTS` to configure JVM settings. The following settings are recommended:

```
CATALINA_OPTS=-server -Xms2048m -Xmx4g
-XX:MaxPermSize=512m
```

Note: If you are deploying multiple Banner 9.x applications to the same Tomcat server, increase the max heap (`-Xmx`) by `2g` and `-XX:MaxPermSize` by `128m`. You should deploy Banner 9.x administrative applications to one Tomcat server instance and Banner 9.x self-service applications to a separate Tomcat server instance.

You can define this variable in the account's profile startup script, or you can add this definition in `$CATALINA_HOME/bin/catalina.sh` for Linux or `catalina.bat` for Windows.

(Optional) If you install Tomcat as a Windows service, specify the JVM arguments as follows:

2.1. Select **Configure Tomcat** application from the Windows **Start** menu.

2.2. Select the **Java** tab.

2.3. In the **Java Options** field, add the following:

```
-XX:MaxPermSize=384m
```

2.4. Set the initial memory pool = 2048.

2.5. Set the maximum memory pool = 4096.

2.6. Save the settings.

2.7. Restart the Tomcat Windows service.

(Optional) To set up the Tomcat server to enable remote JMX connections, perform the steps in the "Configure Java Management Extensions" section. This is useful for debugging and logging.

Define the JNDI datasource resource name for the application as follows:

Edit `$CATALINA_HOME/conf/context.xml`.

Uncomment `<Manager pathname="" />` to disable Tomcat session persistence. For example, change the following:

```
<!-- Uncomment this to disable session persistence across
Tomcat restarts -->

<!--
<Manager pathname="" />
-->
to:
<!-- Uncomment this to disable session persistence across
Tomcat restarts -->

<Manager pathname="" />
```

2.8. Add the following ResourceLink definitions inside the `<Context>` element:

```
<ResourceLink global="jdbc/bannerDataSource"
              name="jdbc/bannerDataSource"
              type="javax.sql.DataSource"/>
<ResourceLink global="jdbc/bannerSsbDataSource"
              name="jdbc/bannerSsbDataSource"
              type="javax.sql.DataSource"/>
```

2.9. Save your changes in `context.xml`.

2.10. Edit `$CATALINA_HOME/conf/server.xml` to configure the database JNDI resource name and connection pool configuration.

2.11. Add the following Resource definitions inside the `<GlobalNamingResources>` element:

- For Tomcat 7

```
<Resource name="jdbc/bannerDataSource" auth="Container"
type="javax.sql.DataSource"
driverClassName="oracle.jdbc.OracleDriver"
url="jdbc:oracle:thin:@//hostname:port/service_name"
username="banproxy" password="the_banproxy_password"
initialSize="5" maxActive="100" maxIdle="-1"
maxWait="30000"
validationQuery="select 1 from dual"
testOnBorrow="true"/>
<Resource name="jdbc/bannerSsbDataSource" auth="Container"
type="javax.sql.DataSource"
driverClassName="oracle.jdbc.OracleDriver"
url="jdbc:oracle:thin:@//hostname:port/service_name"
username="ban_ss_user" password="ban_ss_user_pasword"
initialSize="5" maxActive="100" maxIdle="-1"
maxWait="30000"
validationQuery="select 1 from dual"
testOnBorrow="true"/>
```

- For Tomcat 8

```
<Resource name="jdbc/bannerDataSource" auth="Container"
type="javax.sql.DataSource"
driverClassName="oracle.jdbc.OracleDriver"
url="jdbc:oracle:thin:@//hostname:port/service_name"
username="banproxy" password="the_banproxy_password"
initialSize="5" maxTotal="100" maxIdle="-1"
maxWaitMillis="30000"
validationQuery="select 1 from dual"
accessToUnderlyingConnectionAllowed="true"
testOnBorrow="true"/>
<Resource name="jdbc/bannerSsbDataSource" auth="Container"
```

```
type="javax.sql.DataSource"
driverClassName="oracle.jdbc.OracleDriver"
url="jdbc:oracle:thin:@//hostname:port/service_name"
username="ban_ss_user" password="ban_ss_user_pasword"
initialSize="5" maxTotal="100" maxIdle="-1"
maxWaitMillis="30000"
validationQuery="select 1 from dual"
accessToUnderlyingConnectionAllowed="true"
testOnBorrow="true"/>
```

For example, if your database server name is `myserver.university.edu` and the Oracle TNS Listener is accepting connections on port 1521 and your database service name is SEED, then the URL is `jdbc:oracle:thin:@//myserver.university.edu:1521/SEED.`

2.12. Modify the connector to support UTF-8 URIs by adding the line `URIEncoding="UTF-8"` to the connector.

An example connector would be:

```
<Connector port="8080" protocol="HTTP/1.1"
connectionTimeout="20000"
redirectPort="8443" URIEncoding="UTF-8"/>
```

2.13. Ensure the connector for the StudentSelfService port allows for UTF-8 encoding by adding the following to the connector note.
`URIEncoding="UTF-8"`

An example connector would be:

```
<Connector port="8080" protocol="HTTP/1.1"
connectionTimeout="20000"
redirectPort="8443" URIEncoding="UTF-8">
```

2.14. Save your changes in `server.xml`.

2.15. Copy the Oracle JDBC jar files (`ojdbc6.jar and xdb6.jar`) from the `StudentSelfService/current/lib` directory to the `$CATALINA_HOME/lib` directory.

2.16. Validate the configuration of the Tomcat server by starting the application server. To accomplish this, perform the following steps:

– Run `$CATALINA_HOME/bin/startup.`

For Linux:

```
cd $CATALINA_HOME
$ bin/startup.sh
```

For Windows:

```
cd %CATALINA_HOME%
> bin\startup.bat
```

– Browse http://servername:<port>.

To override the configuration that was added into the WAR file, you must set system properties to point to external configuration files. For example, to point to a configuration

file residing in the PRODUCT_HOME directory, export JAVA_OPTS= "-DBANNER_APP_CONFIG=/PRODUCT_HOME/shared_configuration/banner_configuration.groovy -DBANNER_STUDENT_SSB_CONFIG=/PRODUCT_HOME/StudentSelfService/current/instance/config/StudentSelfService_configuration.groovy -DBANNER_STUDENT_UI_CONFIG=<abolute path to the file bannerStudentAdvisorUI_configuration.properties>".

Configure Java Management Extensions

This is an optional step that is needed only if you want to monitor or debug the application. Java Management Extensions (JMX) is a Java technology that supplies tools for managing and monitoring applications, system objects, devices, and service oriented networks.

Enabling JMX connections allows you to remotely monitor and debug the application server. To enable Java Management Extensions, perform the following steps:

1. Add the following options to the `catalina.sh` or `catalina.bat` file and then restart the Tomcat server:

   ```
   set CATALINA_OPTS=-Dcom.sun.management.jmxremote
   -Dcom.sun.management.jmxremote.port=8999
   -Dcom.sun.management.jmxremote.ssl=false
   -Dcom.sun.management.jmxremote.authenticate=false
   -Djava.rmi.server.hostname=your.hostname.com
   ```

2. Change the `java.rmi.server.hostname` value to the hostname or IP address of the machine where Tomcat is installed. For example:

   ```
   -Djava.rmi.server.hostname=prod.appserver1.com
   ```

   or

   ```
   -Djava.rmi.server.hostname=149.24.3.178
   ```

3. JMX does not define a default port number to use. If necessary, change `com.sun.management.jmxremote.port=8999`.

   Note: It is recommended that you connect remotely to the Tomcat server using JMX.

   Warning! *Ensure that the* `jmxremote.authenticate` *parameter is not set to* `false` *in a production environment. If it is set to* `false`*, it does not require connections to be authenticated and will create a security threat in a production environment. For more information on Tomcat Remote JMX documentation, see http://tomcat.apache.org/tomcat-6.0-doc/monitoring.html#Enabling_JMX_Remote.*

Deploy the WAR file to the Tomcat server

The systool that is used to create the WAR file can also be used to deploy the WAR file to a Tomcat container. You should deploy 9.x administrative applications and 9.x self-service applications to separate Tomcat servers to increase performance.

Note: The systool does not provide the capability to undeploy or redeploy an application. If you are redeploying the application, you must use the Tomcat Manager web application to undeploy the existing application.

The target supports deploying the `dist/WAR` file using the Tomcat Manager web application. Because environments vary significantly with respect to user privileges, clustering approach, web container version, operating system, and more, the target may or may not be suitable for your use.

Note: You can also deploy the WAR file to the Tomcat server by copying the WAR file to the Tomcat `webapps/` directory.

To use the target, you must provide the following information:

URL	This is the URL of the manager application in the Tomcat server. For example: `http://localhost:8080/manager`
Username	This Tomcat server user name must have privileges to deploy WAR files.
Password	This is the password of the Tomcat server user.

Username/password combinations are configured in your Tomcat user database `<TOMCAT_HOME>\conf\tomcat-users.xml.` For Tomcat 7x and 8x, you must configure at least one username/password combination with the manager role. For example:

```
<user username="tomcat" password="tomcat" (your password)
roles="manager-gui, manager"/>
```

Note: The roles in Tomcat server changed between point releases in version 7x and 8x. Refer to the Tomcat documentation specific to your release for information on enabling access to provide the appropriate role to a user account for deployment.

To deploy the WAR file to the Tomcat server, perform the following steps:

1. Navigate to the `StudentSelfService\current\installer` directory.
2. Enter one of the following commands:

   On Unix:
   ```
   $ bin/systool deploy-tomcat
   ```

   On Windows:
   ```
   > bin\systool deploy-tomcat
   ```
3. Enter the following URL for the Tomcat Manager:

   ```
   []: http://localhost:8080/manager
   ```

   This URL will be accessed to deploy the WAR file into the container.
4. Enter a valid Tomcat username to deploy the WAR file. For example:

   ```
   []: tomcat
   ```

   Note: This user must have the `manager-gui` role.
5. Enter the Tomcat password for the user:

   ```
   []: password
   ```

   Note: This password will not be persisted.
6. Access the web application:

   ```
   http://servername:<port>/StudentSelfService
   ```

WebLogic

The following sections provide information on configuring the web application and deploying the WAR file to the WebLogic server:

Note: If you choose to install the application on a WebLogic server, you do not need to install it on Tomcat.

Verify WebLogic prerequisites

Before configuring your WebLogic server, ensure that the following prerequisites are met:

- WebLogic must be installed. If it is not, download and install WebLogic from the Oracle web site.
- The minimum requirements are OFM 11.1.1.4 using WebLogic 10.3.4.
- Both the WebLogic node manager and the administration server must be started. The administration server can be accessed using the following URL:

  `http://server:7001/console`

Set up the cookie path

For a WebLogic installation, the cookie path needs to match the location where the application is deployed. Otherwise, the cookies will not be found by the application. If this change is not made, users will be prompted to log in each time they switch between applications.

Add the following in the `weblogic.xml` file:

```
<wls:session-descriptor>
<wls:cookie-path>/<WebApp_Root_Context></wls:cookie-path>
</wls:session-descriptor>
```

Here are sample steps for the Banner Student Self-Service application.

1. Enter `jar xvf StudentSelfService.war WEB-INF/weblogic.xml`
2. Enter `vi WEB-INF/weblogic.xml`
3. Add the code listed above for the cookie path to the appropriate place in the file. (See the example that follows.)
4. Enter `jar uvf StudentSelfService.war WEB-INF/weblogic.xml`
5. Redeploy the application.

Here is an example of the file that is created.

```
<?xml version="1.0" encoding="UTF-8"?>
<wls:weblogic-web-app
xmlns:wls="http://www.bea.com/ns/weblogic/weblogic-web-app"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://
java.sun.com/xml/ns/javaee/web-app_2_5.xsd http://www.bea.com/ns/
weblogic/weblogic-web-app http://www.bea.com/ns/weblogic/weblogic-web-
app/1.0/weblogic-web-app.xsd">
<wls:weblogic-version>10.3.0</wls:weblogic-version>
<wls:container-descriptor>
<wls:prefer-web-inf-classes>true</wls:prefer-web-inf-classes>
<wls:show-archived-real-path-enabled>true</wls:show-archived-real-path-
enabled>
</wls:container-descriptor>
<wls:session-descriptor>
<wls:cookie-path>/StudentSelf-Service</wls:cookie-path>
</wls:session-descriptor>
</wls:weblogic-web-app>
```

Create a WebLogic machine

Note: If you previously created a WebLogic machine definition, you can skip this section.

To create a WebLogic machine, perform the following steps:

1. In the Change Center frame, click **Lock & Edit**.
2. In the Domain Structure frame, click (**+**) to expand and view the list of environments.
3. Click the **Machines** link.
4. Click **New**.
5. Enter a machine name and click **Next**.
6. Accept the defaults and click **Finish**.
7. In the Change Center frame, click **Activate Changes**.

Create a WebLogic server

Note: If you previously created a WebLogic server, you can skip this section.

Note: If you previously created a WebLogic server for the application, you can use the same server.

To create a WebLogic server, perform the following steps:

1. In the Change Center frame, click **Lock & Edit**.
2. In the Domain Structure frame, click (**+**) to expand and view the list of environments.
3. Click the **Servers** link.

4. Click **New**.
5. Enter a server name and server listen port. For example, you can have server name as `Banner9-SS` and server listen port as `8180`.
6. Click **Finish**.
7. Click the newly created server link.
8. Under the **General** tab, assign the machine to this server.
9. Click **Save**.
10. Select the **Server Start** tab.
11. Add the following to the **Arguments** text area:

    If you are using Sun JVM, use the following parameters:

    ```
    -server -Xms2048m -Xmx4g -XX:MaxPermSize=512m
    ```

    Note: If you are deploying multiple Banner 9.x applications to the same WebLogic server, increase the max heap (`-Xmx`) by `2g` and `-XX:MaxPermSize` by `128m`. You should deploy Banner 9.x administrative applications to one WebLogic server instance and Banner 9.x self-service applications to a separate WebLogic server instance.

    To override the configuration that was added into the WAR file, you can set system properties to point to external configuration files. Append the following to the arguments text area:

    ```
    -DBANNER_APP_CONFIG=<full file path to
    banner_configuration.groovy>
    -DBANNER_STUDENT_SSB_CONFIG=<full file path to
    StudentSelfService_configuration.groovy>
    -DBANNER_STUDENT_UI_CONFIG=<abolute path to the file
    bannerStudentAdvisorUI_configuration.properties>
    ```
12. Click **Save**.
13. In the Change Center frame, click **Activate Changes**.
14. In the Domain Structure frame, click the **Servers** link.
15. Select the **Control** tab.
16. Select the check box next to your new server definition.
17. Click **Start**.

Configure weblogic.xml file to make Banner 9.x JSession cookie secure

Note: This configuration is only specific to WebLogic Server.

Add the following configuration information to the `weblogic.xml` file to make the Banner 9.x JSession cookie secure.

```
<wls:session-descriptor>
<wls:cookie-secure>true</wls:cookie-secure>
<wls:url-rewriting-enabled>false</wls:url-rewriting-enabled>
</wls:session-descriptor>
```

Note: The configuration changes should be added based on specifications at your institution. Once the cookie has been secured, the same application cannot be accessed through a non-SSL port. These configuration changes should only be applied if the SSL is in use.

Update Oracle JDBC JAR files on the WebLogic server

1. Copy the Oracle JAR files (ojdbc6.jar and xdb6.jar) from the $PRODUCT_HOME/current/lib directory to the $MIDDLEWARE_HOME/modules directory.
   - $PRODUCT_HOME is where the Self-Service release zip file is unpacked and installed.
   - $MIDDLEWARE_HOME is the location where Oracle WebLogic is installed.
2. For Linux/Unix servers, edit the setDomainEnv.sh file under the $MIDDLEWARE_HOME/user_projects/domains/<CUSTOM_DOMAIN>/bin folder and add these two lines after the ADD EXTENSIONS comment as shown by the example below:

   #ADD EXTENSIONS TO CLASSPATH

   export MIDDLEWARE_HOME=/u01/app/oracle/Middleware

   export WLS_MODULES=${MIDDLEWARE_HOME}/modules

   export EXT_PRE_CLASSPATH=${WLS_MODULES}/

   xdb6.jar:${WLS_MODULES}/ojdbc6.jar

   Note: If you plan to "copy and paste" the configuration settings into the "setDomainEnv.sh" file, make sure there is no typo or special characters that get carried over (especially with double quotes on the variable declarations). If you see "Class NotFoundException" in your logs, chances are there was a typo when you edited the "setDomainEnv.sh" file and the "xdb6.jar" or "ojdbc6.jar" file cannot be found during Application startup.

3. For MS Windows servers, edit the setDomainEnv.cmd under the $MIDDLEWARE_HOME/user_projects/domains/<CUSTOM_DOMAIN>/bin folder and add these two lines after the ADD EXTENSIONS comment as shown by the example below:

   @REM ADD EXTENSIONS TO CLASSPATH

   set MIDDLEWARE_HOME=D:\Oracle\Middleware

set WLS_MODULES=%MIDDLEWARE_HOME%\modules

set

EXT_PRE_CLASSPATH=%WLS_MODULES%\xdb6.jar;%WLS_MODULES%\ojdbc6.jar

Note: If you plan to "copy and paste" the configuration settings into the "setDomainEnv.cmd" file, make sure there is no typo or special characters that get carried over (especially with double quotes on the variable declarations). If you see "Class NotFoundException" in your logs, chances are there was a typo when you edited the "setDomainEnv.cmd" file and the "xdb6.jar" or "ojdbc6.jar" file cannot be found during Application startup.

4. Restart the WebLogic Managed Server.

Create an administrative datasource and connection pool

Note: If you previously created an administrative datasource, you can skip this section.

To create an administrative datasource and connection pool, perform the following steps:

1. In the Change Center frame, click **Lock & Edit**.
2. In the Domain Structure frame, click (**+**) to expand Services and then select **Data Sources**.
3. Click **New**.
4. Select **Generic DataSource**.
5. Specify a name (for example, `Banner9DS`).
6. Specify the JNDI name (for example, `jdbc/bannerDataSource`).
7. Specify `Oracle` for Database Type and then click **Next**.
8. Select **Oracle Driver (Thin) for Service Connections** and then click **Next**.
9. Clear the **Supports Global Transactions** check box and then click **Next**.
10. Enter the database name, host name, port, user name, password, and password confirmation, and then click **Next**. For example:

Database name:	BAN9
Host name:	`yourhostname.yourdomain.com`
Port:	`1521`
UserName:	`banproxy`
Password:	`your_password`

11. Click **Test Configuration**.
12. Click **Next** for the connection test to be successful.
13. Select the server that you previously created to allow the datasource to be deployed and used by this server.
14. Click **Finish**.
15. Select the datasource link that you created.
16. Select the **Connection Pool** tab.
    16.1. Set the Initial Capacity parameter to specify the minimum number of database connections to create when the server starts up. For example:

    ```
    Initial Capacity = 5
    ```

    16.2. Set the Maximum Capacity parameter to specify the maximum number of database connections that can be created. For example:

    ```
    Maximum Capacity = 100
    ```

17. Change `Statement Cache Type = Fixed`.
18. Change `Statement Cache Size = 0`.
19. Click **Save**.
20. In the Change Center frame, click **Activate Changes**.

Create a self-service datasource and connection pool

Note: If you previously created a self-service datasource, you can skip this section.

To create a self-service datasource and connection pool, perform the following steps:

1. In the Change Center frame, click **Lock & Edit**.
2. In the Domain Structure frame, click (**+**) to expand Services and then select **Data Sources**.
3. Click **New**.
4. Select **Generic DataSource**.
5. Specify a name (for example, `Banner9SsbDS`).
6. Specify the JNDI name (for example, `jdbc/bannerSsbDataSource)`.
7. Specify `Oracle` for Database Type and then click **Next**.
8. Select **Oracle Driver (Thin) for Service Connections** and then click **Next**.
9. On the Transaction Options page, clear the **Supports Global Transactions** check box and then click **Next**.

10. Enter the database name, host name, port, user name, password, and password confirmation, and then click **Next**. For example:

Database name:	`BAN9`
Host name:	`yourhostname.yourdomain.com`
Port:	`1521`
UserName:	`ban_ss_user`
Password:	`your_password`

11. Click **Test Configuration**.
12. Click **Next** for the connection test to be successful.
13. Select the server that you previously created to allow the datasource to be deployed and used by this server.
14. Click **Finish**.
15. Select the datasource link that you created.
16. Select the **Connection Pool** tab.
    - **16.1.** Set the Initial Capacity parameter to specify the minimum number of database connections to create when the server starts up. For example:

      ```
      Initial Capacity = 5
      ```

    - **16.2.** Set the Maximum Capacity parameter to specify the maximum number of database connections that can be created. For example:

      ```
      Maximum Capacity = 100
      ```

17. Change `Statement Cache Type = LRU`.
18. Change `Statement Cache Size = 20`.
19. Click **Save**.
20. In the Change Center frame, click **Activate Changes**.

Configure server communication

When using different servers for administrative and self-service applications, perform the following steps:

1. In the Change Center frame, click **Lock & Edit**.
2. In the Domain Structure frame, click (**+**) to expand Services and then select **DataSources**.
3. Select the datasource for the administrative application (for example, `Banner9DS`).
4. Select the **Targets** tab.
5. Select the check box for the self-service server (for example, `Banner9-SS`).

6. Click **Save**.
7. In the Change Center frame, click **Activate Changes**.

Deploy and start the application in the WebLogic server

To deploy and start the web application in the WebLogic server, perform the following steps:

1. Change the name of the WAR file to remove the version number. For example, change:

   ```
   StudentSelfService/current/dist/StudentSelfService-9.7.war
   ```

   to

   ```
   StudentSelfService/current/dist/StudentSelfService.war
   ```

2. Access the administration server at the following URL:

   ```
   http://server:7001/console
   ```

3. In the Domain Structure frame, select the **Deployments** link.
4. In the Change Center frame, select **Lock and Edit**.
5. Click **Install**.
6. Select the WAR file to be deployed and then click **Next**. The file is located in the following directory:

   ```
   StudentSelfService/current/dist
   ```

7. Select **Install this deployment** as an application and then click **Next**.
8. Select the target server on which to deploy this application (for example, `Banner9-SS`) and then click **Next**.
9. Click **Finish**.
10. In the Change Center frame, click **Activate Changes**.
11. Select the deployed application and then click **Start**.
12. Select **Servicing all request**.
13. Access the application using the following URL format:

    ```
    http://servername:<port>/<web application>
    ```

    For example:

    ```
    http://localhost:8080/StudentSelfService
    ```

14. Log in to the application using a valid username and password.

Configure the application

The following aspects of the application can be configured.

Name format

Banner General 8.8.5 provides the foundation of a new approach to formatting names. Self-Service pages can now build on this foundation for the Self-Service portion of the application.

Institutions must definitely configure the name format and adjust them as desired as mentioned in the Name Display (GUANDSP) form.

The following are the default values that must not be changed by the institutions.

```
// preferred name's default values

productName = "Student"

banner.applicationName = "Student Self-Service"
```

For more information about name formats, refer to the Name Functionality section of the *Banner Student Self-Service Handbook*. You can find the handbook in the Documentation Libraries in the Ellucian Support Center.

Phone format

Within the `StudentSelfService/current/i18n` folder, find the `message.properties` file for your language. The default for American English is the `message.properties` file. Open the file with a text editor. Add an entry for the `default.personTelephone.format` setting, using the following elements:

Area code:	`$phoneArea`
Phone number:	`$phoneNumber`
Extension:	`$phoneExtension`
Telephone country code:	`$phoneCountry`
International access code:	`$phoneInternational`

Date format

The date format can be customized by using the following keys in the `messages_<ISO_language_code>_<ISO_country_code>.properties` file in the `StudentSelfService\current\i18n` directory:

- `default.date.format`
- `js.datepicker.dateFormat`

default.date.format

This key determines the date format for display and data entry in the user interface. It must match the ICU specification and can be changed based on locale. For more information, refer to the following URL: http://userguide.icu-project.org/formatparse/datetime.

For the `default.date.format` for June 1, 2014, use one of the following variables for the year:

Year format	Interpretation	Comment
yy	14	Two digit year
yyyy	2014	Four digit year

For the `default.date.format` for June 1, 2014, use one of the following variables for the month:

Month format	Interpretation	Comment
M	6	Single digit month (no leading zero)
MM	06	Double digit month
MMM	Jun	Short month name
MMMM	June	Long month name

For the `default.date.format` for June 1, 2014, use one of the following variables for the day:

Day format	Interpretation	Comment
d	1	Single digit day in a month (no leading zero)
dd	01	Double digit day in a month

The `default.date.format` and `js.datepicker.dateFormat` formats use different specifications to represent the date. The ICU format is dd/MM/yyyy. The Java Script or Keith Wood format is dd/mm/yyyy. For dates to be displayed properly, the two formats must match. For example, for 1 June, 2012, the calendar parses 01/06/2012 using

`js.datepicker.dateFormat`, and when the dates are saved, the date value on the server side is parsed by groovy using `default.date.format` to display 01/06/2012 in the application.

js.datepicker.dateFormat

This key determines the date format for the interactive date selection control. It must match the jQuery Keith Wood datepicker format specification. For more information on the Keith Wood datepicker format, refer to the following URL: http://keith-wood.name/datepick.html#format.

For the `js.datepicker.dateFormat` for June 1, 2014, use one of the following variables for the year:

Year format	Interpretation	Comment
yy	14	Two digit year
yyyy	2014	Four digit year

For the `js.datepicker.dateFormat` for June 1, 2014, use one of the following variables for the month:

Month format	Interpretation	Comment
m	6	Single digit month (no leading zero)
mm	06	Double digit month
M	Jun	Short month name
MM	June	Long month name

For the `js.datepicker.dateFormat` for June 1, 2014, use one of the following variables for the day:

Day format	Interpretation	Comment
d	1	Single digit day in a month (no leading zero)
dd	01	Double digit day in a month

Time format

The `default.time.format` is a simple Java date format key available in the Banner Student Self-Service application. For more information on the Java date format, refer to http://docs.oracle.com/javase/6/docs/api/java/text/SimpleDateFormat.html.

Multiple calendars

Customization of multiple calendars is implemented for the Arabic language (AR). The file name for Arabic is `messages_ar.properties`.

To display multiple calendars in the application, you must use the following keys:

- `default.calendar`
- `default.calendar1`
- `default.calendar2`

Note: The `default.calendar2` is optional.

In the following example, the keys set your default calendar format as Islamic, the first alternate calendar displayed as Gregorian, and the second alternate calendar as Arabic.

```
default.calendar=islamic

default.calendar1=gregorian

default.calendar2=islamic
```

Depending on the order you want to view the calendars, you can interchange the following values:

- `islamic`
- `gregorian`

CSS customization

To customize the appearance of the self-service application, you can provide custom CSS and image files.

Create a CSS file `StudentSelfService/current/instance/css/bannerSelfService-custom.css` containing the custom CSS directives.

If you want to provide custom images, save them in the `StudentSelfService/current/instance/css/images` directory, and in the CSS, specify their paths as a URL. For example:

```
(./images/filename.png)
```

Note: If this directory structure does not exist, create the directory structure.

Institution name

The default layout includes an institutional branding area that displays the institution name or logo. The default layout styles the institutional branding area as follows:

```
.institutionalBranding {
    position:relative;
    float:left;
    left:10px;
    top:11px;
    height:19px;
    width:179px;
    background:url("images/ellucian-university-logo-sm.png") no-repeat;}
```

To customize the system or university name, you must provide a replacement logo image and a custom CSS file to override the default styling. To override the image, create a CSS file named `StudentSelfService/current/instance/css/bannerSelfService-custom.css` that contains the following:

```
.institutionalBranding {background-image: url("./images/
institutionLogo.png");}
```

Place the logo image in the following directory:

```
StudentSelfService/current/instance/css/images/institutionLogo.png
```

To deploy your updates, you must rebuild and redeploy the WAR file. For more information, see “Regenerate the WAR file” on page 47.

Custom JavaScript

You can optionally add a custom JavaScript file by placing your JavaScript file named `bannerSelfService-custom.js` in the following location:

```
StudentSelfService/current/instance/js/bannerSelfService-custom.js
```

Install Banner Student Self-Service Application for SAML 2.0 SSO

The following sections detail the installation steps for the Banner Student Self-Service application with the Ellucian Ethos Identity as the primary Identity Management System that supports the SAML 2.0 SSO protocol:

- "Undeploy the existing application" on page 68
- "Customize the WAR file" on page 70
- "Generate SAML 2.0 metadata files" on page 74
- "Configure application-specific settings" on page 77
- "Set up SAML 2.0 SSO Configuration" on page 79
- "Customize the landing page background image" on page 83
- "Regenerate the WAR file" on page 85
- "Configure and deploy the WAR file to a web application server" on page 86
- "Add service provider in Ellucian Ethos Identity Server" on page 96
- "Modify Identity Provider Issuer" on page 98
- "SAML 2.0 Configuration Sub-tasks" on page 99

Undeploy the existing application

Before you reinstall Banner Student Self-Service 9.7, you must undeploy the previous 9.x versions of Banner Student Self-Service, Banner Student Attendance Tracking Self-Service (if installed), and Banner Student Advisor Self-Service (if installed). If no previous 9.x versions of Banner Student Self-Service, Student Attendance Tracking Self-Service, or Banner Student Advisor Self-Service are installed, you can skip this section.

Tomcat

You can either use the Tomcat Manager web application to undeploy the existing application or you can shut down Tomcat and manually remove the files.

Undeploy using the Tomcat Manager web application

Use the following procedure to undeploy the application using the Tomcat Manager web application:

1. Access the Tomcat Manager web application at one of the following URLs:

   ```
   http://server:8080/manager
   - or -
   http://server:8080/manager/html
   ```

2. Access the deployment page using a valid user name and password.
3. Under the Commands area, click **Stop** to stop the existing application.
4. In the confirmation dialog box, click **OK**.
5. Under the Commands area, click **Undeploy**.
6. In the confirmation dialog box, click **OK** to undeploy the application.

Undeploy using a manual procedure

The following sections give the steps to manually undeploy the existing application on Unix and Windows operating systems.

Unix

Use the following procedure to manually undeploy the existing application on a Unix operating system:

1. Log in to the server where Tomcat is running, using the same account credentials that were used to start Tomcat.
2. Shut down Tomcat by running the shutdown script:

   ```
   $CATALINA_HOME/bin/shutdown.sh
   ```

3. Remove the current deployment and associated WAR file:

   ```
   cd $CATALINA_HOME

   rm -rf $CATALINA_HOME/webapps/StudentSelfService

   rm -rf $CATALINA_HOME/webapps/StudentSelfService.war
   ```

4. If you have a previous version Student Attendance Tracking (9.2 or older) installed, remove the deployment and associated WAR file.
5. If you have a previous version Student Advisor (9.3.0.2 or older) installed, remove the deployment and associated WAR file.

Windows

Use the following procedure to manually undeploy the existing application on a Windows operating system:

1. Use the command prompt to shut down Tomcat.

Note: If you installed Tomcat as a service, use the Service Control panel to stop the application. Otherwise, use the shutdown script `%CATALINA_HOME%\bin\shutdown.bat`.

2. Remove the current deployment and associated WAR file:

   ```
   rmdir %CATALINA_HOME%\webapps\StudentSelfService /s/q
   ```

   ```
   del %CATALINA_HOME%\webapps\StudentSelfService.war /q
   ```

3. If you have a previous version Student Attendance Tracking (9.2 or older) installed, remove the deployment and associated WAR file.
4. If you have a previous version Student Advisor (9.3.0.2 or older) installed, remove the deployment and associated WAR file.

WebLogic

Use the following procedure to stop and undeploy the application:

1. Access the administration server using the following URL:

   ```
   http://server:7001/console
   ```

2. In the Domain Structure frame, click **Deployments**.
3. In the Change Center, click **Lock and Edit**.
4. Select the check box to the left of the application.
5. Click **Stop**.
6. Click **Force Stop Now**.
7. In the Force Stop Application Assistant page, click **Yes**.
8. Select the check box to the left of the application.
9. Click **Delete**.
10. In the Delete Application Assistant page, click **Yes**.
11. In the Change Center frame, click **Activate Changes**.

Customize the WAR file

The name of the release package is `release-StudentSelfService-9.7.zip`. This release package is moved to the `$BANNER_HOME\student\java` subdirectory during the database upgrade. Use the following steps to unzip the release package and customize the WAR file for your institution.

Note: JDK 1.7 must be installed on your system. See the Java dependencies section for more information.

When you locate the release package zip file, copy it to your application server environment. Transfer this file in binary mode using File Transfer Protocol (FTP). To copy the release package, you must have a valid application server account.

Unzip the release package

To unzip the release package into a temporary directory, perform the following steps:

1. Log in to the application server platform.

   Note: You must have a valid application server account to deploy into the application server container (Tomcat or WebLogic).

2. Create a temporary directory. For example:

   ```
   mkdir $HOME/ban9temp
   ```

3. Locate the release package `release-StudentSelfService-9.7.zip.`
4. Transfer this file in binary mode using File Transfer Protocol (FTP) file into the temporary directory. For example:

   ```
   $HOME/ban9temp
   ```

5. Unzip `release-StudentSelfService-9.7.zip` into the temporary directory.

Prepare the installer

To prepare the installer, perform the following steps:

1. Change the directory to the installer directory:

   ```
   cd installer
   ```

2. Run the `ant` command, which will build the installation tool.

   Note: For Unix, make sure the ant file is executable. For example, `chmod +x ant`.

Example:

```
ban9temp $ cd installer
ban9temp/installer $ ./ant
```

The message *Build successful* confirms a successful build.

Install into the product home directory

The product home directory supports the configuration and creation of a deployable WAR file. Although Banner 9.x web applications are modular and are installed independently, they share a common configuration. The package provides a common installer that creates consistent product home directory structures for all Banner 9.x applications.

Within a particular environment, you should place the product home directories for Banner 9.x applications in sibling directories. For example, the following directory structure includes four product home directories and a `shared_configuration` directory that support a common test environment.

```
banner_test_homes
|--> Registration 9.2
|--> Catalog 9.3
|--> Schedule 9.3
|--> applicationNavigator
|--> shared_configuration
```

A product home directory is created for each deployment. For example, the home directory that is used for the application within a test environment is different than the home directory that is used for the production environment. When you are supporting different environments for multiple home directories for the same solution, this structure provides the necessary configuration, release level, and custom modification flexibility.

The following directory tree illustrates the product home directory that is created for the test environment:

```
TEST/                                  (optional and recommended top-level directory for all homes)

|-->app-name                           (product home for 'app-name' in test environment)
    |--> current
        |--> instance/                 (instance-specific configuration that will not be overwritten)
            |--> config/
                |--> {app-name}_configuration.groovy (module-specific configuration for CAS, logging, etc.)
            |--> i18n                  (new or replacement message bundles that should be added the war)
            |--> css                   (new or replacement css files that should be added the war)
            |--> js                    (new or replacement javascript files that should be added the war)

        |--> lib
            |--> ojdbc6.jar           (the Oracle database driver that must be placed manually into the tomcat/lib directory)
            |--> logging.properties   (logging configuration that may be copied to the WEB-INF/classes directory that is
                                       very useful if the war file cannot be deployed successfully.)
        |--> i18n/                     (contains message bundles that may reflect changes not yet in 'baseline')
        |--> dist/                     (contains the war file, after it is creating using the 'systool')
        |--> installer/                (contains the installer)
    |--> archived-releases/            (directory for previous releases)
        |-->

|--> shared_configuration/             (home for configuration files shared across modules within an environment)
    |--> banner_configuration.groovy   (a 'shared  configuration file containing dataSource)
```

In addition to the application's product home directory, a separate `shared_configuration` home directory contains cross-application configuration for the test environment. This directory holds the `banner_configuration.groovy` file, which contains the shared JNDI datasource configuration.

To install the installer into the product home directory, perform the following steps:

1. Ensure that the installer is prepared using `ant`.
2. Use the installer to install the release file into the product home directory.

Note: Your current working directory must be in the installer directory (`ban9temp/installer`) before executing the following commands.

On Unix:

```
$ bin/install home
```

On Windows:

```
> bin\install home
```

3. When prompted, enter the full path of the application home directory. The application will be installed within the current subdirectory within this home directory and the previous release will be archived.

   On Unix:

   ```
   []: Current_home_directory/banner_test_homes/
   StudentSelfService
   ```

   On Windows:

   ```
   []: c:\banner_test_homes\StudentSelfService
   ```

4. Enter the full path of the `shared_configuration` home directory. Banner 9.x applications that refer to this home directory share this configuration file.

   On Unix:

   ```
   []: Current_home_directory/banner_test_homes/
   shared_configuration
   ```

   On Windows:

   ```
   []: c:\banner_test_homes\shared_configuration
   ```

Note: If an identified home directory or the `shared_configuration` home directory does not exist, the installer creates it. The name of a product home directory is not restricted. You can name it when prompted by the installer.

Configure shared settings

The `shared_configuration` home directory contains a cross-application configuration file called `banner_configuration.groovy`. You can change settings in this file.

JNDI datasource

You can optionally change the datasource name in the configuration file to point to the JNDI datasource that is configured in your application server. For example, `jndiName = "jdbc/bannerSsbDataSource"` is the default configuration. You can change this to match the JNDI datasource name in your environment.

Generate SAML 2.0 metadata files

This section discusses the creation and handling of metadata files. The following files must be created:

- keystore file
- service provider file
- identity service provider file

An IDP Certificate entry must be added in the newly created keystore file. Then the keystore, service provider, and identity service provider files must be added to the WAR file creation location.

Create a keystore (*.jks) file

1. Create a keystore file (`*.jks`) with the name used in step 2.

   See "Create a keystore (*.jks) file" on page 99 for more information.

2. Place this file in the location specified in the following key value:
   "`grails.plugin.springsecurity.saml.keyManager.storeFile = 'classpath:security/studentkeystore.jks'`"

Create a service provider file

1. Create a file `banner-<short-appName>-sp.xml` at the folder path mentioned in "SAML 2.0 SSO Configuration" on page 80.

2. Edit the `banner-<short-appName>-sp.xml` file for the service provider configuration which will configure the Authentication end point and Logout endpoint.

   Replace the parameters below with the configured values for Banner Student Self-Service.

   - <HOSTNAME>: Application host name
   - <PORT>: Deployed Application port number
   - <ALIAS_NAME>: Service provider ID set in Ellucian Ethos Identity service provider setup
   - <EXTRACTED_DATA>: Extract a X509 Certificate key from the keystore for `banner-<short-appName>-sp.xml` (See "Extract a X509 Certificate Key" on page 100 for more information.)

     Place the extracted value in the `banner-<short-appName>-sp.xml` file:

     `banner-<short-appName>-sp.xml`

   Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<md:EntityDescriptor xmlns:md="urn:oasis:names:tc:SAML:2.0:metadata"
ID="<ALIAS_NAME>" entityID="<ALIAS_NAME>">
<md:SPSSODescriptor AuthnRequestsSigned="false" WantAssertionsSigned="false"
protocolSupportEnumeration="urn:oasis:names:tc:SAML:2.0:protocol">
<md:KeyDescriptor use="signing">
<ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
<ds:X509Data>
<ds:X509Certificate>
<EXTRACTED_DATA>
</ds:X509Certificate>
</ds:X509Data>
</ds:KeyInfo>
</md:KeyDescriptor>
<md:KeyDescriptor use="encryption">
<ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
<ds:X509Data>
<ds:X509Certificate>
<EXTRACTED_DATA>
</ds:X509Certificate>
</ds:X509Data>
</ds:KeyInfo>
</md:KeyDescriptor>
<md:SingleLogoutService Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST"
Location="http://<HOSTNAME>:<PORT>/<APPLICATION_NAME>/saml/SingleLogout/alias/
<ALIAS_NAME>"/>
<md:SingleLogoutService Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-
Redirect" Location="http://<HOSTNAME>:<PORT>/<APPLICATION_NAME>/saml/SingleLogout/
alias/<ALIAS_NAME>"/>
<md:NameIDFormat>urn:oasis:names:tc:SAML:1.1:nameid-format:emailAddress</
md:NameIDFormat>
<md:NameIDFormat>urn:oasis:names:tc:SAML:2.0:nameid-format:transient</
md:NameIDFormat>
<md:NameIDFormat>urn:oasis:names:tc:SAML:2.0:nameid-format:persistent</
md:NameIDFormat>
<md:NameIDFormat>urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified</
md:NameIDFormat>
<md:NameIDFormat>urn:oasis:names:tc:SAML:1.1:nameid-format:X509SubjectName</
md:NameIDFormat>
<md:AssertionConsumerService Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-
POST" Location="http://<HOSTNAME>:<PORT>/<APPLICATION_NAME>/saml/SSO/alias/
<ALIAS_NAME>" index="0" isDefault="true"/>
<md:AssertionConsumerService Binding="urn:oasis:names:tc:SAML:2.0:profiles:holder-
of-key:SSO:browser" Location="http://<HOSTNAME>:<PORT>/<APPLICATION_NAME>/saml/
SSO/alias/<ALIAS_NAME>"
hoksso:ProtocolBinding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Artifact"
index="1" xmlns:hoksso="urn:oasis:names:tc:SAML:2.0:profiles:holder-of-
key:SSO:browser"/>
<md:AssertionConsumerService Binding="urn:oasis:names:tc:SAML:2.0:profiles:holder-
of-key:SSO:browser" Location="http://<HOSTNAME>:<PORT>/<APPLICATION_NAME>/saml/
SSO/alias/<ALIAS_NAME>"
hoksso:ProtocolBinding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST" index="2"
xmlns:hoksso="urn:oasis:names:tc:SAML:2.0:profiles:holder-of-key:SSO:browser"/>
</md:SPSSODescriptor>
</md:EntityDescriptor>
```

Create an identity service provider file

1. Create a file `banner-<short-appName>-idp.xml` at the folder path mentioned in "SAML 2.0 SSO Configuration" on page 80.
2. Edit the `banner-<short-appName>-idp.xml` file for the identity provider configuration.

   The file contains the identity provider information configured in Ellucian Ethos Identity server over which Banner Student Self-Service sends the SAMLrequest and receives the SAML response.

   Replace below parameters with the configured values for Banner Student Self-Service.

   - <HOSTNAME>: Ellucian Ethos Identity server host name
   - <PORT>: Deployed Ellucian Ethos Identity server port number
   - <EXTRACTED_DATA>: Extract X509 certificate Data for `banner-<short-appName>-idp.xml` (See "Extract X509 certificate data" on page 102 for more information.)

     Place the extracted value in the `banner-<short-appName>-idp.xml` file.

     `banner-<short-appName>-idp.xml`

   Example:

```
<?xml version="1.0"?>
<md:EntityDescriptor xmlns:md="urn:oasis:names:tc:SAML:2.0:metadata"
entityID="https://<HOSTNAME>:<PORT>/samlsso" cacheDuration="PT1440M">
<md:IDPSSODescriptor
protocolSupportEnumeration="urn:oasis:names:tc:SAML:2.0:protocol">
<md:KeyDescriptor use="signing">
<ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
<ds:X509Data>
<ds:X509Certificate>
<EXTRACTED_DATA>
</ds:X509Certificate>
</ds:X509Data>
</ds:KeyInfo>
</md:KeyDescriptor>
<md:KeyDescriptor use="encryption">
<ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
<ds:X509Data>
<ds:X509Certificate>
<EXTRACTED_DATA>
</ds:X509Certificate>
</ds:X509Data>
</ds:KeyInfo>
</md:KeyDescriptor>
<md:SingleLogoutService Location="https://<HOSTNAME>:<PORT>/samlsso"
Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Redirect"/>
<md:SingleLogoutService Location="https://<HOSTNAME>:<PORT>/samlsso"
Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST"/>
```

```
<md:NameIDFormat>urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified</
md:NameIDFormat>
<md:SingleSignOnService Location="https://<HOSTNAME>:<PORT>/samlsso"
Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST"/>
<md:SingleSignOnService Location="https://<HOSTNAME>:<PORT>/samlsso"
Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Redirect"/>
</md:IDPSSODescriptor>
<md:ContactPerson contactType="administrative"/>
</md:EntityDescriptor>
```

Add an IDP Certificate entry in the newly created keystore file

Add the IDP certificate entry in the newly created `.jks` file. See "Add IDP certificate entry to the .jks file" on page 103 for more information.

Add keystore, service provider, and identity service provider files to WAR file creation location

Place the three files created in this section into the `StudentSelfService\current\instance\config` directory.

After adding the files, the directory should contain the following:

- `studentkeystore.jks` (The newly created `.jks` file)
- `banner-<short-appName>-idp.xml`
- `banner-<short-appName>-sp.xml`
- `StudentSelfService_configuration.groovy`

Configure application-specific settings

The `StudentSelfService\current\instance\config` directory contains the `StudentSelfService_configuration.groovy` file. This application-specific configuration file contains settings that you can customize for your specific environment. This directory also contains an `instance.properties` file that references the shared configuration location.

Self-service end point

The `ssbEnabled` setting is set to `true` for instances that expose self-service end points. Banner Student Self-Service is an administrative application, so the default value is always `false`.

JMX MBean name

The name that is used to register MBeans must be unique for each application that is deployed into the JVM. This configuration should be updated for each instance of each application to ensure uniqueness.

```
jmx {
    exported {
        log4j = "student-self-service-log4j"
    } }
```

Location of the logging file

Log4j is the common logging framework used with applications that run on the Java Virtual Machine. For more information, refer to the log4j documentation.

The configuration file includes documentation on various elements that can be modified depending on your environment.

The following is an example of how to override the location where the log file is saved.

```
loggingFileDir = "System.properties['logFileDir'] ?
"${System.properties['logFileDir']}" : "target/logs"
logAppName = "StudentSelfService"
loggingFileName = "${loggingFileDir}/${logAppName}.log".toString()
```

The following is an example of how to override the log file directory properties:

```
export JAVA_OPTS = "-DlogFileDir=/PRODUCT_HOME /"
```

The output logging file location is relative to the application server to which you are deploying.

Logging level

The root logging level is pre-configured to the ERROR level. Multiple class or package level configurations, by default, are set to a status of "off." You can set a different logging level for any package or class. However, the running application must be restarted.

For example:

```
case 'production':
root {
error 'appLog' //change the log level here with the
appropriate log level value.
additivity = true
}
```

Note: Changing the logging level to DEBUG or INFO produces very large log files.

Changes to the `StudentSelfService_configuration.groovy` file require a restart of the application before those changes take effect.

Alternatively, you can use JMX to modify logging levels for any specified package or class, or even at the root level. When using JMX, the logging level changes only affect the running application. When you restart the application, changes that you made using JMX are lost.

For more information on JMX configuration, see "Configure Java Management Extensions" on page 88.

Set up SAML 2.0 SSO Configuration

The `StudentSelfService\current\instance\config` directory contains the `StudentSelfService_configuration.groovy` file. This application specific configuration file contains settings that you can customize for your specific environment.

Authentication Provider Name

The name identifying the application authentication mechanism. Values are **cas** or **saml**. Specify saml to indicate the application will use SAML 2.0 SSO protocol authentication as shown in the following example:

```
/*****************************************************
*     BANNER AUTHENTICATION PROVIDER CONFIGURATION    *
*                                                     *
*****************************************************/
//
// Set authenticationProvider to either default, cas or saml.
// If using cas or saml, Either the CAS CONFIGURATION or the SAML
CONFIGURATION
// will also need configured/uncommented as well as set to active.
//
banner {
    sso {
       authenticationProvider = 'default' //  Valid values are: 'saml' and
       'cas' for SSO to work. 'default' to be used only for zip file
       creation.
       authenticationAssertionAttribute = 'UDC_IDENTIFIER'
      if(authenticationProvider != 'default') {
      grails.plugin.springsecurity.failureHandler.defaultFailureUrl = '/
      login/error'
      }
      if(authenticationProvider == 'saml') {
      grails.plugin.springsecurity.auth.loginFormUrl = '/saml/login'
      }
```

```
    }
}
```

Logout URL

You can specify where a user is directed after logging out of the application by updating the `StudentSelfService_configuration.groovy` file. There are three ways the application can handle logouts:

- Logouts can display the CAS logout page with a redirect URL.
- Logouts can automatically go to a redirect URL (without displaying the CAS logout page).
- Logouts can take the user to a custom logout page with a redirect URL to Home Page.

In SAML mode, logout directs the user to a custom logout page.

SAML 2.0 SSO Configuration

Shown below is a sample of the configuration you can enable for SSO between Banner Student Self-Service and an Identity Management System that support SAML 2.0 SSO protocol. Make sure to uncomment this section when SAML 2.0 SSO is enabled.

The following properties describe the configurations required for the application to work in SAML 2.0 SSO protocol. Each property is described in the following table:

Property	Description
`banner.sso.authentication.saml.localLogout`	Default value is set to false, indicating that the application will participate in Global logout. An application participating in global logout will notify the Identity Server about logout within the application. If this is set to true, indicating local logout, the application will not notify the Identity Server to log the user out from all applications.
`grails.plugin.springsecurity.auth.loginFormUrl`	Pre-populated to provide the Login URL.
`grails.plugin.springsecurity.saml.afterLogoutUrl`	Pre-populated to provide the Logout URL.
`grails.plugin.springsecurity.saml.keyManager.defaultKey`	Key name used at the time of key-store creation ("Create a keystore (*.jks) file" on page 99). Example: `studentkeystore.jks`

Property	Description
`grails.plugin.springsecurity.saml.keyManager.storeFile`	Location of the keystore file. This could be a classpath (for example, `classpath:securitystudentkeystore.jks`) or it could be an absolute location on the machine (for example, `file:c://temp/studentkeystore.jks` or `u02/studentkeystore.jks`).
`grails.plugin.springsecurity.saml.keyManager.storePass`	Password used to decrypt the keys in the keystore created above.
`grails.plugin.springsecurity.saml.keyManager.passwords`	Key value pair to validate the key. Contains the alias key name used at the time of key-store creation and password used to decrypt the key.
`grails.plugin.springsecurity.saml.metadata.sp.file`	Location of the service provider metadata file. This could be a classpath location, (for example, `security/sp.xml`) or it could be a absolute location on the machine (for example, `C://temp/banner-student-sp file:/home/u02/banner-sp.xml`).
`grails.plugin.springsecurity.saml.metadata.providers`	Key value pair mapping to validate the identity provider configured in `banner-<short-appName>-idp.xml`. Example: `adfs : 'security/banner-student-idp.xml'`. Possible keys are adfs, Okta, Shibb, etc.
`grails.plugin.springsecurity.saml.metadata.defaultIdp`	Provide the default IDP to be used from the IDP providers set. This is the same value specified above for the key.

Property	Description
grails.plugin.springsecurity.saml.metadata.sp.defaults	• local: Pre-populated to indicate value to be picked up. • alias: An alias name that is unique to this application (for example, banner-<application-shortname>-sp). • securityProfile: Pre-populated value. • signingKey: A key used to sign the messages that is unique to this application (for example, banner-<application-shortname>-sp). • encryptionKey: A key to to encrypt the message that is unique to this application, (for example, banner-<application-shortname>-sp) • tlsKey: A tls key that is unique to this application (for example, banner-<application-shortname>-sp). • requireArtifactResolveSigned: Pre-Populated to set to false indicating artifact to be signed or not. • requireLogoutRequestSigned: Pre-Populated to set to false indicating logout request to be signed or not. • requireLogoutResponseSigned: Pre-Populated to set to false indicating logout response to be signed or not.

```
/*********************************************************
*                                                        *
*                SAML CONFIGURATION                      *
* Uncomment this section if Banner Student Advisor is configured with SAML
*********************************************************//*

// set active = true when authentication provider section configured for
saml

/*

grails.plugin.springsecurity.saml.active = false

grails.plugin.springsecurity.auth.loginFormUrl = '/saml/login'

grails.plugin.springsecurity.saml.afterLogoutUrl ='/logout/customLogout'

banner.sso.authentication.saml.localLogout='false'

// To disable single logout set this to true,default 'false'.

grails.plugin.springsecurity.saml.keyManager.storeFile =
'classpath:security/<KEY_NAME>.jks'

// for unix File based Example:- 'file:/home/u02/samlkeystore.jks'

grails.plugin.springsecurity.saml.keyManager.storePass = 'test1234'

grails.plugin.springsecurity.saml.keyManager.passwords = [ 'banner-
<short-appName>-sp': 'test1234']

// banner-<short-appName>-sp is the value set in Ellucian Ethos Identity
Service provider setup

grails.plugin.springsecurity.saml.keyManager.defaultKey = 'banner-<short-
appName>-sp'// banner-<short-appName>-sp is the value set in Ellucian
Ethos Identity Service provider setup


grails.plugin.springsecurity.saml.metadata.sp.file = 'security/banner-
<Application_Name>-sp.xml'// for unix file based Example:-'/home/u02/sp-
local.xml'
```

```
grails.plugin.springsecurity.saml.metadata.providers = [adfs: 'security/
banner-<Application_Name>-idp.xml'] // for unix file based Example: '/
home/u02/idp-local.xml'

grails.plugin.springsecurity.saml.metadata.defaultIdp =
'adfs'grails.plugin.springsecurity.saml.metadata.sp.defaults =


       [local: true,

       alias: 'banner-<short-appName>-sp',

       // banner-<short-appName>-sp is the value set in Ellucian Ethos
       Identity Service provider setup securityProfile: 'metaiop',

       signingKey: 'banner-<short-appName>-sp',// banner-<short-appName>-
       sp is the value set in Ellucian Ethos Identity Service provider
       setup

       encryptionKey: 'banner-<short-appName>-sp', // banner-<short-
       appName>-sp is the value set in Ellucian Ethos Identity Service
       provider setup

       tlsKey: 'banner-<short-appName>-sp',// banner-<short-appName>-sp is
       the value set in Ellucian Ethos Identity Service provider setup

       requireArtifactResolveSigned: false,

       requireLogoutRequestSigned: false,

       requireLogoutResponseSigned: false

]
```

Customize the landing page background image

To customize and replace the landing page background image with an institutional image of your choice, you must perform the following steps before building and deploying the WAR file to the Application Server:

1. Create two CSS files in the deployment staging directory of Banner Student Self-Service (example: `~/StudentSelfService/current/instance/css`).
   - `bannerSelfService-custom.css` for LTR support
   - `bannerSelfService-custom-rtl.css` file for RTL support
2. Modify each of these two files by adding the custom CSS styles you wish to change as shown below. For overriding the landing page background image, use the following CSS class and add your institutional background image to the media query style that supports the specific responsive design orientation.

```
/** Custom CSS file which takes precedence over the landing page CSS styles **/


.landing-content {
    margin-left: 0px;
    padding-left: 0px;
    background-repeat: no-repeat;
    background-position:center;
    background-size: cover;
    background-color: #4F585F;
}


@media (orientation:landscape) {
```

```
.landing-content {
background-image: url("/css/images/backgrounds/campus-landscape.jpg");
}
}

@media (orientation:portrait) {
.landing-content {
background-image: url("/css/images/backgrounds/campus-portrait.jpg");
}
}
@media (orientation:portrait) and (max-width:320px) {
.landing-content {
background-image: url("/css/images/backgrounds/campus-sml.jpg");
}
}
```

3. The above images and styles shown support responsive designs for landscape, portrait and mobile views. Make sure your institutional background images you choose maintain the correct aspect ratio for displaying the custom images in landscape and portrait mode along with supporting different device screen resolutions.

   - `campus-landscape.jpg` supports widescreen desktop devices (supported resolution size 1920 * 1080)
   - `campus-portrait.jpg` supports laptops and tablet devices (supported resolution size 768 * 1024)
   - `campus-sml.jpg` supports mobile devices (supported resolution size 320 * 568)

4. Confirm the custom CSS files and images are in the deployment staging directory of Banner Student Self-Service. The directory and file structure should be similar as shown below:

```
StudentSelfService
|-----current
    |-----instance
        |-----css
            |-----bannerSelfService-custom.css
            |-----bannerSelfService-custom-rtl.css
        |-----images
            |-----campus-landscape.jpg
            |-----campus-portrait.jpg
            |-----campus-sml.jpg
```

Note: The image name can be any custom image name. However, css file names must be the same as specified (`bannerSelfService-custom.css` and `bannerSelfService-custom-rtl.css`), and the class name used to override the background image of landing page also must be the same landing-content.

5. Continue with the next steps of regenerating and deploying the Banner Student Self-Service WAR file to your Application Server with the customized CSS files. Verify the landing page back-ground image has changed and meets your institutional needs.

Regenerate the WAR file

This section describes how you can regenerate the application WAR file to include your customizations and application-specific settings. You can then deploy the WAR file into your application server.

Use the systool to create the WAR file. Complete the following steps to set up the systool to create the WAR file:

1. Change your current working directory to the product home directory:

   ```
   PRODUCT_HOME/current/installer
   ```

2. Run the ant command, which builds the systool module.

   For Unix, make sure the ant file is executable, for example, `chmod +x ant.`

   ```
   $ cd PRODUCT_HOME/current/installer
   ```

   ```
   PRODUCT_HOME/current/installer $ ./ant
   ```

3. Create the WAR file using the systool module.

   Your current working directory must be in the `PRODUCT_HOME/current/installer` directory before you execute one of the following commands based on your platform:

   On Unix:

   ```
   $ bin/systool war
   ```

   On Windows:

   ```
   > bin\systool war
   ```

   The WAR file is created in the `PRODUCT_HOME/current/dist directory`.

You can use external configuration files by setting appropriate system properties, although the configuration files are included in the WAR file to make the WAR file self-sufficient. For information on external configuration, see "Configure the Tomcat server" on page 86 or "Create a WebLogic server" on page 91.

Note: If you are using an application that links to the Student application (such as the Student Profile page), there is a different default root context. You will need to change the configuration of those applications to reflect this change. If you will be doing new installations of applications that link to the Student application, please note the new URL context which might not be reflected in the documentation for those applications.

Configure and deploy the WAR file to a web application server

The following sections provides information on configuring the web application and deploying the WAR file to a web application server:

- "Tomcat" on page 86
- "WebLogic" on page 91

Tomcat

The following sections provide information on configuring the web application and deploying the WAR file to the Tomcat server.

Note: If you choose to install the application on a Tomcat server, you do not need to install it on WebLogic.

Configure the Tomcat server

Use the following steps to configure the Tomcat server:

1. Locate the Oracle JDBC jar files (`ojdbc6.jar and xdb6.jar`) in the `PRODUCT_HOME\current\lib` directory.

   Note: Later in the Tomcat configuration process, you will copy the Oracle JDBC jar file into the `\lib` folder under the Tomcat installation directory.

   The account that runs the Tomcat application server must configure environment settings to support the application.

2. On Linux, ensure `CATALINA_HOME` is defined to reference your Tomcat software installation location. For example, `CATALINA_HOME=/opt/apache-tomcat-7.0.xx` where xx indicates the point version of Tomcat you installed.

   Warning! *Do not perform this step on the Windows platform.*

3. Define `CATALINA_OPTS` to configure JVM settings. The following settings are recommended:

   ```
   CATALINA_OPTS=-server -Xms2048m -Xmx4g
   -XX:MaxPermSize=512m
   ```

   Note: If you are deploying multiple Banner 9.x applications to the same Tomcat server, increase the max heap (`-Xmx`) by `2g` and `-XX:MaxPermSize` by `128m`. You should deploy Banner 9.x

administrative applications to one Tomcat server instance and Banner 9.x self-service applications to a separate Tomcat server instance.

You can define this variable in the account's profile startup script, or you can add this definition in `$CATALINA_HOME/bin/catalina.sh` for Linux or `catalina.bat` for Windows.

4. (Optional) If you install Tomcat as a Windows service, specify the JVM arguments as follows:

   4.1. Select **Configure Tomcat** application from the Windows **Start** menu.

   4.2. Select the **Java** tab.

   4.3. In the **Java Options** field, add the following:

   ```
   -XX:MaxPermSize=384m
   ```

   4.4. Set the initial memory pool = 2048.

   4.5. Set the maximum memory pool = 4096.

   4.6. Save the settings.

   4.7. Restart the Tomcat Windows service.

5. (Optional) To set up the Tomcat server to enable remote JMX connections, perform the steps in the "Configure Java Management Extensions" section. This is useful for debugging and logging.

6. Define the JNDI datasource resource name for the application as follows:

   6.1. Edit `$CATALINA_HOME/conf/context.xml`.

   6.2. Uncomment `<Manager pathname="" />` to disable Tomcat session persistence. For example, change the following:

   ```
   <!-- Uncomment this to disable session persistence
   across Tomcat restarts -->

   <!--
   <Manager pathname="" />
   -->
   ```

   to:

   ```
   <!-- Uncomment this to disable session persistence
   across Tomcat restarts -->

   <Manager pathname="" />
   ```

   6.3. Add the following ResourceLink definitions inside the `<Context>` element:

   ```
   <ResourceLink global="jdbc/bannerDataSource"
                 name="jdbc/bannerDataSource"
                 type="javax.sql.DataSource"/>
   ```

   6.4. Save your changes in `context.xml`.

   6.5. Edit `$CATALINA_HOME/conf/server.xml` to configure the database JNDI resource name and connection pool configuration.

6.6. Add the following Resource definitions inside the <GlobalNamingResources> element:

```
<Resource name="jdbc/bannerDataSource" auth="Container"
  type="javax.sql.DataSource"
  driverClassName="oracle.jdbc.OracleDriver"
  url="jdbc:oracle:thin:@//hostname:port/service_name"
  username="banproxy" password="the_banproxy_password"
  initialSize="5" maxActive="100" maxIdle="-1" maxWait="30"
  validationQuery="select 1 from dual"
  testOnBorrow="true"/>
```

For example, if your database server name is myserver.university.edu and the Oracle TNS Listener is accepting connections on port 1521 and your database service name is SEED, then the URL is jdbc:oracle:thin:@//myserver.university.edu:1521/SEED.

6.7. Save your changes in server.xml.

6.8. Copy the Oracle JDBC jar files (ojdbc6.jar and xdb6.jar) from the PRODUCT_HOME/current/lib directory to the $CATALINA_HOME/lib directory.

6.9. Validate the configuration of the Tomcat server by starting the application server. To accomplish this, perform the following steps:

– Run $CATALINA_HOME/bin/startup.

For Linux:

```
cd $CATALINA_HOME
$ bin/startup.sh
```

For Windows:

```
cd %CATALINA_HOME%
> bin\startup.bat
```

– Browse http://servername:<port>.

To override the configuration that was added into the WAR file, you must set system properties to point to external configuration files. For example, to point to a configuration file residing in the PRODUCT_HOME directory, export JAVA_OPTS= "-DBANNER_APP_CONFIG=/PRODUCT_HOME/shared_configuration/banner_configuration.groovy -DBANNER_STUDENT_SSB_CONFIG=/PRODUCT_HOME/StudentSelfService/current/instance/config/StudentSelfService_configuration.groovy -DBANNER_STUDENT_UI_CONFIG=<abolute path to the file bannerStudentAdvisorUI_configuration.properties>".

Configure Java Management Extensions

This is an optional step that is needed only if you want to monitor or debug the application. Java Management Extensions (JMX) is a Java technology that supplies tools for

managing and monitoring applications, system objects, devices, and service oriented networks.

Enabling JMX connections allows you to remotely monitor and debug the application server. To enable Java Management Extensions, perform the following steps:

1. Add the following options to the `catalina.sh` or `catalina.bat` file and then restart the Tomcat server:

   ```
   set CATALINA_OPTS=-Dcom.sun.management.jmxremote
   -Dcom.sun.management.jmxremote.port=8999
   -Dcom.sun.management.jmxremote.ssl=false
   -Dcom.sun.management.jmxremote.authenticate=false
   -Djava.rmi.server.hostname=your.hostname.com
   ```

2. Change the `java.rmi.server.hostname` value to the hostname or IP address of the machine where Tomcat is installed. For example:

   ```
   -Djava.rmi.server.hostname=prod.appserver1.com
   ```

   - or -

   ```
   -Djava.rmi.server.hostname=149.24.3.178
   ```

3. JMX does not define a default port number to use. If necessary, change `com.sun.management.jmxremote.port=8999`.

   Note: It is recommended that you connect remotely to the Tomcat server using JMX.

   Warning! *Ensure that the* `jmxremote.authenticate` *parameter is not set to* `false` *in a production environment. If it is set to* `false`*, it does not require connections to be authenticated and will create a security threat in a production environment.*

Deploy the WAR file to the Tomcat server

The systool that is used to create the WAR file can also be used to deploy the WAR file to a Tomcat container. You should deploy 9.x administrative applications and 9.x self-service applications to separate Tomcat servers to increase performance.

Note: The systool does not provide the capability to undeploy or redeploy an application. If you are redeploying the application, you must use the Tomcat Manager web application to undeploy the existing application.

The target supports deploying the `dist/WAR` file using the Tomcat Manager web application. Because environments vary significantly with respect to user privileges, clustering approach, web container version, operating system, and more, the target may or may not be suitable for your use.

Note: You can also deploy the WAR file to the Tomcat server by copying the WAR file to the Tomcat `webapps/` directory.

To use the target, you must provide the following information:

URL	This is the URL of the manager application in the Tomcat server. For example: `http://localhost:8080/manager`
Username	This Tomcat server username must have privileges to deploy WAR files.
Password	This is the password of the Tomcat server user.

Username/password combinations are configured in your Tomcat user database `<TOMCAT_HOME>\conf\tomcat-users.xml.` For Tomcat 7x and 8x, you must configure at least one username/password combination with the manager role. For example:

```
<user username="tomcat" password="tomcat" (your password)
roles="manager-gui, manager"/>
```

To deploy the WAR file to the Tomcat server, perform the following steps:

1. Navigate to the `PRODUCT_HOME\current\installer` directory.
2. Enter one of the following commands:

   On Unix:
   ```
   $ bin/systool deploy-tomcat
   ```

   On Windows:
   ```
   > bin\systool deploy-tomcat
   ```
3. Enter the following URL for the Tomcat Manager:

   ```
   []: http://localhost:8080/manager
   ```

   This URL will be accessed to deploy the WAR file into the container.
4. Enter a valid Tomcat username to deploy the WAR file. For example:

   ```
   []: tomcat
   ```

   Note: This user must have the `manager-gui` role.
5. Enter the Tomcat password for the user:

   ```
   []: password
   ```

   Note: This password will not be persisted.
6. Access the web application:

   ```
   http://servername:<port>/StudentSelfService
   ```

WebLogic

The following sections provide information on configuring the web application and deploying the WAR file to the WebLogic server:

Note: If you choose to install the application on a WebLogic server, you do not need to install it on Tomcat.

Verify WebLogic prerequisites

Before configuring your WebLogic server, ensure that the following prerequisites are met:

- WebLogic must be installed. If it is not, download and install WebLogic from the Oracle web site.
- The minimum requirements are OFM 11.1.14 using WebLogic 10.3.4.
- Both the WebLogic node manager and the administration server must be started. The administration server can be accessed using the following URL:

  ```
  http://server:7001/console
  ```

Create a WebLogic machine

Note: If you previously created a WebLogic machine definition, you can skip this section.

To create a WebLogic machine, perform the following steps:

1. In the Change Center frame, click **Lock & Edit**.
2. In the Domain Structure frame, click (**+**) to expand and view the list of environments.
3. Click the **Machines** link.
4. Click **New**.
5. Enter a machine name and click **Next**.
6. Accept the defaults and click **Finish**.
7. In the Change Center frame, click **Activate Changes**.

Create a WebLogic server

Note: If you previously created a WebLogic server, you can skip this section.

Note: If you previously created a WebLogic server for the application, you can use the same server.

To create a WebLogic server, perform the following steps:

1. In the Change Center frame, click **Lock & Edit**.
2. In the Domain Structure frame, click (**+**) to expand and view the list of environments.
3. Click the **Servers** link.
4. Click **New**.
5. Enter a server name and server listen port. For example, you can have server name as `Banner9-SS` and server listen port as `8180`.
6. Click **Finish**.
7. Click the newly created server link.
8. Under the **General** tab, assign the machine to this server.
9. Click **Save**.
10. Select the **Server Start** tab.
11. Add the following to the **Arguments** text area:

    If you are using Sun JVM, use the following parameters:

    ```
    -server -Xms2048m -Xmx4g -XX:MaxPermSize=512m
    ```

    Note: If you are deploying multiple Banner 9.x applications to the same WebLogic server, increase the max heap (`-Xmx`) by `2g` and `-XX:MaxPermSize` by `128m`. You should deploy Banner 9.x administrative applications to one WebLogic server instance and Banner 9.x self-service applications to a separate WebLogic server instance.

    To override the configuration that was added into the WAR file, you can set system properties to point to external configuration files. Append the following to the arguments text area:

    ```
    -DBANNER_APP_CONFIG=<full file path to
    banner_configuration.groovy>
    -DBANNER_STUDENT_SSB_CONFIG=<full file path to
    StudentSelfService_configuration.groovy>
    -DBANNER_STUDENT_UI_CONFIG=<abolute path to the file
    bannerStudentAdvisorUI_configuration.properties>
    ```

12. Click **Save**.
13. In the Change Center frame, click **Activate Changes**.
14. In the Domain Structure frame, click the **Servers** link.
15. Select the **Control** tab.
16. Select the check box next to your new server definition.

17. Click **Start**.

Update Oracle JDBC JAR files on the WebLogic server

1. Copy the Oracle JAR files (ojdbc6.jar and xdb6.jar) from the $PRODUCT_HOME/current/lib directory to the $MIDDLEWARE_HOME/modules directory.
   - $PRODUCT_HOME is where the Self-Service release zip file is unpacked and installed.
   - $MIDDLEWARE_HOME is the location where Oracle WebLogic is installed.
2. For Linux/Unix servers, edit the setDomainEnv.sh file under the $MIDDLEWARE_HOME/user_projects/domains/<CUSTOM_DOMAIN>/bin folder and add these two lines after the ADD EXTENSIONS comment as shown by the example below:

   ```
   #ADD EXTENSIONS TO CLASSPATH

   export MIDDLEWARE_HOME=/u01/app/oracle/Middleware

   export WLS_MODULES=${MIDDLEWARE_HOME}/modules

   export EXT_PRE_CLASSPATH="${WLS_MODULES}/

   xdb6.jar:${WLS_MODULES}/ojdbc6.jar
   ```

   Note: If you plan to "copy and paste" the configuration settings into the "setDomainEnv.sh" file, make sure there is no typo or special characters that get carried over (especially with double quotes on the variable declarations). If you see "Class NotFoundException" in your logs, chances are there was a typo when you edited the "setDomainEnv.sh" file and the "xdb6.jar" or "ojdbc6.jar" file cannot be found during Application startup.

3. For MS Windows servers, edit the setDomainEnv.cmd under the $MIDDLEWARE_HOME/user_projects/domains/<CUSTOM_DOMAIN>/bin folder and add these two lines after the ADD EXTENSIONS comment as shown by the example below:

   ```
   @REM ADD EXTENSIONS TO CLASSPATH

   set MIDDLEWARE_HOME=D:\Oracle\Middleware

   set WLS_MODULES=%MIDDLEWARE_HOME%\modules

   set

   EXT_PRE_CLASSPATH=%WLS_MODULES%\xdb6.jar;%WLS_MODULES%\ojdbc6
   .jar
   ```

   Note: If you plan to "copy and paste" the configuration settings into the "setDomainEnv.cmd" file, make sure there is no typo or special characters that get carried over (especially with double quotes on the variable declarations). If you see "Class NotFoundException" in your logs, chances are there was a typo when you edited the "setDomainEnv.cmd"

file and the "xdb6.jar" or "ojdbc6.jar" file cannot be found during Application startup.

4. Restart the WebLogic Managed Server.

Create an administrative datasource and connection pool

Note: If you previously created an administrative datasource, you can skip this section.

To create an administrative datasource and connection pool, perform the following steps:

1. In the Change Center frame, click **Lock & Edit**.
2. In the Domain Structure frame, click (**+**) to expand Services and then select **Data Sources**.
3. Click **New**.
4. Select **Generic DataSource**.
5. Specify a name (for example, `Banner9DS`).
6. Specify the JNDI name (for example, `jdbc/bannerDataSource)`.
7. Specify `Oracle` for Database Type and then click **Next**.
8. Select **Oracle Driver (Thin) for Service Connections** and then click **Next**.
9. Clear the **Supports Global Transactions** check box and then click **Next**.
10. Enter the database name, host name, port, user name, password, and password confirmation, and then click **Next**. For example:

Database name:	BAN9
Host name:	`yourhostname.yourdomain.com`
Port:	`1521`
UserName:	`banproxy`
Password:	`your_password`

11. Click **Test Configuration**.
12. Click **Next** for the connection test to be successful.
13. Select the server that you previously created to allow the datasource to be deployed and used by this server.
14. Click **Finish**.
15. Select the datasource link that you created.
16. Select the **Connection Pool** tab.

16.1. Set the Initial Capacity parameter to specify the minimum number of database connections to create when the server starts up. For example:

```
Initial Capacity = 5
```

16.2. Set the Maximum Capacity parameter to specify the maximum number of database connections that can be created. For example:

```
Maximum Capacity = 100
```

17. Change `Statement Cache Type = Fixed`.
18. Change `Statement Cache Size = 0`.
19. Click **Save**.
20. In the Change Center frame, click **Activate Changes**.

Deploy and start the application in the WebLogic server

To deploy and start the web application in the WebLogic server, perform the following steps:

1. Change the name of the WAR file to remove the version number. For example, change:

   ```
   StudentSelfService/current/dist/StudentSelfService-
   9.7.war
   ```

   to

   ```
   StudentSelfService/current/dist/StudentSelfService.war
   ```

2. Access the administration server at the following URL:

   ```
   http://server:7001/console
   ```

3. In the Domain Structure frame, select the **Deployments** link.
4. In the Change Center frame, select **Lock and Edit**.
5. Click **Install**.
6. Select the WAR file to be deployed and then click **Next**. The file is located in the following directory:

   ```
   StudentSelfService/current/dist
   ```

7. Select **Install this deployment** as an application and then click **Next**.
8. Select the target server on which to deploy this application (for example, `Banner9-SS`) and then click **Next**.
9. Click **Finish**.
10. In the Change Center frame, click **Activate Changes**.
11. Select the deployed application and then click **Start**.
12. Select **Servicing all request**.
13. Access the application using the following URL format:

```
http://servername:<port>/<web application>
```

For example:

```
http://localhost:8080/StudentSelfService
```

14. Log in to the application using a valid username and password.

Add service provider in Ellucian Ethos Identity Server

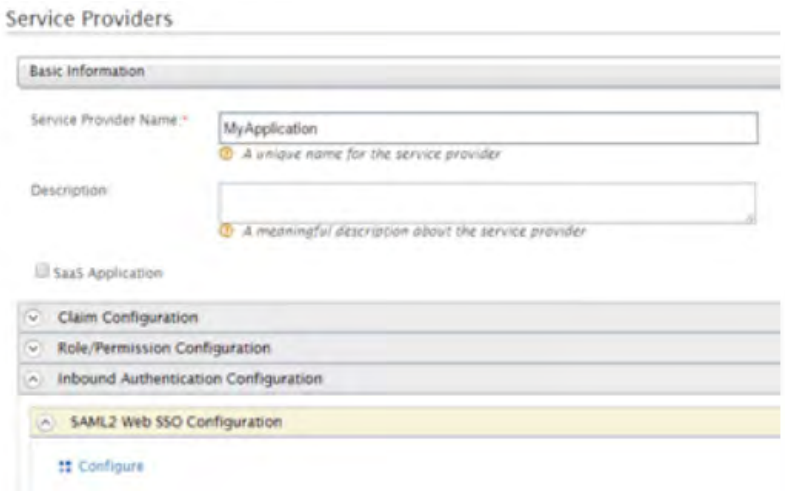
To add a service provider, complete the following steps:

1. Log in to the Ellucian Ethos Identity Management Console and click the **Main** tab.
2. Under Service Providers, click **Add**. The **Add Service Provider** screen appears.
3. Enter a service provider name in the **Service Provider Name** field. You can also add an optional description in the **Description** field.
4. Click **Register** to add the service provider. The **Service Providers** screen appears.

Add SAML settings

To add SAML setting for the integrating application, complete these steps:

1. On the Service Providers screen, expand the Inbound Authentication Configuration panel, then expand the SAML2 Web SSO Configuration panel, as shown in the following example:



2. Click the **Configure** link.
3. Enter the Issuer value that is configured in the service provider application. This value is validated against the SAML Authentication Request issued by the service provider.

   Note: Make sure to provide the same value that is configured as the "alias" in the configuration property `'grails.plugin.springsecurity.saml.metadata.sp.defaults'` in the `StudentSelfService_configuration.groovy` file.

4. Enter a valid Assertion Consumer URL where the browser redirects the SAML Response after authentication.
5. Enter a valid NameID format supported by Ellucian Ethos Identity. The following values can be used.
   - `urn:oasis:names:tc:SAML:2.0:nameid-format:persistent`
   - `urn:oasis:names:tc:SAML:2.0:nameid-format:transient`
   - `urn:oasis:names:tc:SAML:1.1:nameid-format:emailAddress`
   - `urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified`
   - `urn:oasis:names:tc:SAML:1.1:nameid-format:X509SubjectName`
   - `urn:oasis:names:tc:SAML:1.1:nameid-format:WindowsDomainQualifiedName`
   - `urn:oasis:names:tc:SAML:2.0:nameid-format:kerberos`
   - `urn:oasis:names:tc:SAML:2.0:nameid-format:entity`
6. Select **Use fully qualified username** in the NameID if the user store domain and user ID must be preserved in the SAML 2.0 Response. In most cases, this can be left unselected.
7. Select **Enable Response Signing** to sign the SAML 2.0 Responses returned after the authentication process.
8. Select **Enable Assertion Signing** to sign the SAML 2.0 Assertions returned after the authentication.
9. Select **Enable Signature Validation** in Authentication Requests and Logout Requests if the identity provider must validate the signature of the SAML 2.0 Authentication and Logout Requests that are sent by the service provider.
10. Select **Assertion Encryption** to encrypt the assertions.
11. Select **Certificate Alias** for the service provider's public certificate.

    This certificate is used to validate the signature of SAML 2.0 Requests and is used to generate encryption.
12. Select **Enable Single Logout** so that all sessions across all authenticated service providers are terminated once the user signs out from one server.

If the service provider supports a different URL than the Assertion Consumer URL for logout, enter a Custom Logout URL for logging out. This should match the URL set up in the `.xml` file property "SingleLogoutService" set in."Create a service provider file" on page 74.

13. Select **Enable Attribute Profile** to add a basic attribute profile where the identity provider can include the user's attributes in the SAML Assertions as part of the attribute statement.
14. Select **Include Attributes** in the Response Always if the identity provider should always include the attribute values related to the selected claims in the SAML attribute statement.

    This is required so that UDC_IDENTIFIER configured in claims are sent across.
15. Select **Enable Audience Restriction** to restrict the audience. Add audience members using the Audience text box and click Add Audience.
16. Select **Enable IdP Initiated SSO** and the service provider is not required to send the SAML 2.0 Request.
17. Click **Register** to save settings and return to the Service Provider Configuration page.
18. Click **Update** to save all settings.

Modify Identity Provider Issuer

This step provides instructions on how to add a resident identity provider as per the `idp-local.xml` configuration.

The Ellucian Ethos Identity Server can mediate authentication requests between service providers and identity providers. At the same time, the identity server can act as a service provider and an identity provider. When acting as an identity provider, it is known as the resident identity provider. This converts the identity server into a federated hub.

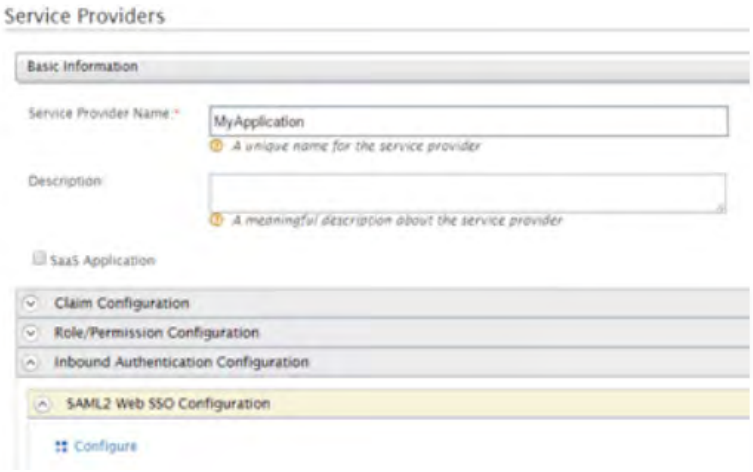
The resident identity provider configuration is relevant for you if you are a service provider and want to send an authentication request or a provisioning request to the identity server (for example, via SAML, OpenID, OpenID Connect, SCIM, and WS-Trust).

Resident identity provider configuration is a one-time configuration for a given tenant. It shows you the identity server's metadata, like the endpoints. In addition, you can secure the WS-Trust endpoint with a security policy.

You must change the Identity Provider Entity Id to the expected URL of the Issuer statement in SAML 2.0 Responses. Complete the following steps to change the ID.

1. Log in to the Ellucian Ethos Identity Management Console and click the **Main** tab.
2. In the Main menu under the Identity section, click **List** under Identity Providers.
3. Enter a service provider name in the **Service Provider Name** field. You can also add an optional description in the **Description** field.
4. Click **List** under Identity Providers.

5. Click **Resident Identity Provider**.
6. Expand the Inbound Authentication Configuration panel, then expand the SAML2 Web SSO Configuration panel, as shown in the following example:



7. Enter a valid identity provider name or URL to be used by all service providers.
8. Click **Update** to save the settings.

SAML 2.0 Configuration Sub-tasks

Subtasks referred to in the Install Banner Student Self-Service Application for SAML 2.0 SSO chapter are discussed in this section.

Create a keystore (*.jks) file

During SAML configuration, you must create a keystore file (`*.jks`). Follow these steps to create the file:

1. Open your operating system's command console and navigate to the directory where `keytool.exe` is located. This is usually where the JRE is located (for example, `c:\Program Files\Java\jre7\bin` on Windows machines).
2. Run the command below (where validity is the number of days before the certificate expires).

   2.1. Fill in the prompts for your organization information.

2.2. When asked for your first and last name, enter the domain name of the server that users will be entering to connect to your application.

```
C:\Program Files\Java\jdk1.7.0_67\jre\bin>keytool -genkey -keyalg RSA -alias mykey
-keystore studentkeystore.jks -storepass password -validity 360 -keysize 2048
What is your first and last name?
  [Unknown]:  John Doe
What is the name of your organizational unit?
  [Unknown]:  Ellucian
What is the name of your organization?
  [Unknown]:  Ellucian
What is the name of your City or Locality?
  [Unknown]:  Malvern
What is the name of your State or Province?
  [Unknown]:  PA
What is the two-letter country code for this unit?
  [Unknown]:  US
Is CN=John Doe, OU=Ellucian, O=Ellucian, L=Malvern, ST=PA, C=US correct?
  [no]:  yes
Enter key password for <mykey>
(RETURN if same as keystore password):changeit
C:\Program Files\Java\jdk1.7.0_67\jre\bin>
```

This creates an `studentkeystore.jks` file containing a private key and your self-signed certificate.

Extract a X509 Certificate Key

During SAML configuration, you must extract a X509 certificate key from the keystore for the `banner-<Application_Name>-sp.xml` file. Follow these steps to extract one.

1. With the `studentkeystore.jks` file you created, execute the command below to check which certificates are in a Java keystore:

See "Create a keystore (*.jks) file" on page 99 for more information.

```
C:\Program Files\Java\jdk1.7.0_67\jre\bin>keytool -list -v -keystore
studentkeystore.jks
Enter keystore password:
Keystore type: JKS
Keystore provider: SUN
Your keystore contains 1 entry
Alias name: mykey
Creation date: Apr 6, 2015
Entry type: PrivateKeyEntry
Certificate chain length: 1
Certificate[1]:
Owner: CN=John Doe, OU=Ellucian, O=Ellucian, L=Malvern, ST=PA, C=US
Issuer: CN=John Doe, OU=Ellucian, O=Ellucian, L=Malvern, ST=PA, C=US
Serial number: 78548b7
Valid from: Mon Apr 06 12:52:39 IST 2015 until: Thu Mar 31 12:52:39 IST 2016
Certificate fingerprints:
```

```
MD5:  5D:55:F4:18:3D:CF:AE:5A:27:B8:85:68:42:47:CA:76
SHA1: CD:09:04:F5:01:60:14:CC:DF:48:07:4A:93:99:17:BF:10:83:F3:55
SHA256:
79:5A:7F:0C:A4:B1:0E:30:9C:B0:DD:87:2C:CA:19:A1:0E:89:29:2F:95:A1:35:E9:EC:A2:AA:B
9:F6:2D:BE:35
Signature algorithm name: SHA256withRSA
Version: 3
Extensions:
#1: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: B2 AC D7 09 01 15 22 A3   32 08 86 64 E8 25 5A 15  ......".2..d.%Z.
0010: CB A0 C6 D9                                        ....
]
]
*******************************************
*******************************************
```

This command shows all the available keystores in the `.jks` file.

Note: To get information about the specific certificate, execute this command:

```
keytool -list -v -keystore studentkeystore.jks -
alias mykey
```

2. Execute the following command to export a certificate from a keystore:

```
C:\Program Files\Java\jdk1.7.0_67\jre\bin>keytool -export
-alias mykey -file mykey.crt -keystore
studentkeystore.jks

Enter keystore password: password

Certificate stored in file <mykey.crt>
```

3. Execute the following command to get the X509 certificate:

```
C:\Program Files\Java\jdk1.7.0_67\jre\bin>keytool -printcert -rfc -file mykey.crt
-----BEGIN CERTIFICATE-----
MIIDfTCCAmWgAwIBAgIEB4VItzANBgkqhkiG9w0BAQsFADBvMQswCQYDVQQGEwJJTjELMAkGA1UE
CBMCSU4xEjAQBgNVBAcTCUJhbmdhbG9yZTERMA8GA1UEChMIRWxsdWNpYW4xETAPBgNVBAsTCEVs
bHVjaWFuMRkwFwYDVQQDExBTcGhvb3J0aSBBY2hhcnlhMB4XDTE1MDQwNjA3MjIzOVoXDTE2MDMz
MTA3MjIzOVowbzELMAkGA1UEBhMCSU4xCzAJBgNVBAgTAklOMRIwEAYDVQQHEwlCYW5nYWxvcmUx
ETAPBgNVBAoTCEVsbHVjaWFuMREwDwYDVQQLEwhFbGx1Y2lhbjEZMBcGA1UEAxMQU3Bob29ydGkg
QWNoYXJ5YTCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBAN2cS+2OX37HioFzwOWLm/S0
F+zt6ldtLfmHc16V9iqkZChkMiXpKgXVPqGLFjBrhwfsWtuMfRy2NYf3forEDFTaV4/fLXRo+Npd
xTfqWhuZTafDJEyKQc57KY8G3feg1CSjfKkCk1LF+zbGClHQ0bg1dwUjJlp7eKjWM0rbsKMd5pZ7
0tGAPcYsi6MtGvJupaVhy3jNTDg+kh4/D92y/mTaLlCR4QQr1qlU9+H+it3m9jiDrZ7svrdBlsDN
1BVcXDooqUGTuc10IBxYEsb7hFucSFpdJnGJvbg35ll9K9F5S8lEmiQmeOUQ1gQe2Ow01kFl56Qz
4evM0xgeskNid9sCAwEAAaMhMB8wHQYDVR0OBBYEFLKs1wkBFSKjMgiGZOglWhXLoMbZMA0GCSqG
SIb3DQEBCwUAA4IBAQANJbYRTcMwhrETz+mo+n1okrXIs118AIm7s1yJJd1nyJuaKrn7DcPPLzy/
RjHGKP02uLiupgqar+UUaPqSZjJXSzktLLyq7H6DRrW0Jp2rw48a+Kou+XOvQ8ZWR9ZXIa1XoAoD
```

```
PaSSE2omcVOVGZmQKUYardVeSvQth3IVMW9w9Jl+DuavXavVjIx5IN6RRhXGfaJjQLKFzIDqZNAp
OcxMEKXHOUqj0ksTRARLpKWSPu7gFOWO/6qapNp5l8rlPjnVxDhqHqCKC3E40VI5n+C+KJHZQqab
Tfhd6erqEy7S1Cazr655Yq22Jm6L7IXsXgpRwmZnoietLsrFIRyPe1DY
-----END CERTIFICATE-----
```

Extract X509 certificate data

During SAML configuration, you must extract X509 certificate data for banner-<Application_Name>-sp.xml. Follow these steps to extract data:

1. Go to the deployed WSO2 server / Ellucian Ethos Identity server. For example, `C:\>cd C:\work\eis\`.
2. Navigate to the server certificate location. By default, it is located at: `$EIS_HOME\repository\resources\security`
3. Execute the following command, where `saml-idp.cer` is the certificate file in the server:

   ```
   C:\work\eis\repository\resources\security>keytool -
   printcert -rfc -file saml-idp.cer
   ```

   This would return a value similar to the following:

```
-----BEGIN CERTIFICATE-----
MIICdDCCAd2gAwIBAgIEEsIIcDANBgkqhkiG9w0BAQsFADBtMRAwDgYDVQQGEwdVbmtub3duMRAw
DgYDVQQIEwdVbmtub3duMRAwDgYDVQQHEwdVbmtub3duMRAwDgYDVQQKEwdVbmtub3duMRAwDgYD
VQQLEwdVbmtub3duMREwDwYDVQQDEwhXU08yIElEUDAeFw0xNDA4MTgxMzA3NTFaFw0xNTA4MTgx
MzA3NTFaMG0xEDAOBgNVBAYTB1Vua25vd24xEDAOBgNVBAgTB1Vua25vd24xEDAOBgNVBAcTB1Vu
a25vd24xEDAOBgNVBAoTB1Vua25vd24xEDAOBgNVBAsTB1Vua25vd24xETAPBgNVBAMTCFdTTzIg
SURQMIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQC44MMcoAPb+aR8l5gtTsSb+SzslHFESmKL
wO1+SAY6p2iiO+G9qR/511ufCzKWrMdMMpoCJLC0myDwoUuGvk0dycTpm5NwUX6CnqDtYhtGkYg8
JT+LtG67k6yjXNa9wrE6VBJynDDPnlL8gLUl9ZCIFrmevJ75rOCaLsoFsghHPwIDAQABoyEwHzAd
BgNVHQ4EFgQULyCNnvW9Ngy5zM7Waf205mR11ZAwDQYJKoZIhvcNAQELBQADgYEApWlSy2GUSaHM
Kkc8XZmdQ0//SId8DKRKaFaZW388K4dJGTSWUnzq4iCWFrAN9O4D1DBnNE+dCDEmV8HvmyQBedsG
JnAre0VisqKz9CjIELcGUaEABKwkOgLe1YyqV29vS4Y3PuTxAhbkyphFb5PxjHDHH/WLQ8pOTbsC
vX4wO04=
-----END CERTIFICATE-----
```

   3.1. If no certificate file is found, navigate to the `.jks` file. The `.jks` file can be found via `carbon.xml`. By default, it is located at:

   `$EIS_HOME\repository\conf\carbon.xml`

   3.2. Locate the KeyStore file location in the `carbon.xml` file, as shown in the following example:

```
<KeyStore>
    <!-- Keystore file location-->
    <Location>${carbon.home}/repository/resources/security/cacerts</Location>
    <!-- Keystore type (JKS/PKCS12 etc.)-->
    <Type>JKS</Type>
    <!-- Keystore password-->
    <Password>changeit</Password>
```

```
    <!-- Private Key alias-->
    <KeyAlias>mykey</KeyAlias>
    <!-- Private Key password-->
    <KeyPassword>changeit</KeyPassword>
</KeyStore>
```

3.3. Create the `saml-idp.cer` file by executing the following command:

```
keytool -export -keystore cacerts -alias mykey -file
~/saml-idp.cer
```

4. Go to the keystore location and execute the following command.

Note: The `password` and `alias` referenced in this example are also contained in the `carbon.xml` file accessed earlier in this task.

```
C:\work\eis\repository\resources\security>keytool -export -keystore cacerts -rfc -
alias mykey
Enter keystore password:
-----BEGIN CERTIFICATE-----
MIICdDCCAd2gAwIBAgIEEsIIcDANBgkqhkiG9w0BAQsFADBtMRAwDgYDVQQGEwdVbmtub3duMRAw
DgYDVQQIEwdVbmtub3duMRAwDgYDVQQHEwdVbmtub3duMRAwDgYDVQQKEwdVbmtub3duMRAwDgYD
VQQLEwdVbmtub3duMREwDwYDVQQDEwhXU08yIElEUDAeFw0xNDA4MTgxMzA3NTFaFw0xNTA4MTgx
MzA3NTFaMG0xEDAOBgNVBAYTB1Vua25vd24xEDAOBgNVBAgTB1Vua25vd24xEDAOBgNVBAcTB1Vu
a25vd24xEDAOBgNVBAoTB1Vua25vd24xEDAOBgNVBAsTB1Vua25vd24xETAPBgNVBAMTCFdTTzIg
SURQMIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQC44MMcoAPb+aR8l5gtTsSb+SzslHFESmKL
wO1+SAY6p2iiO+G9qR/511ufCzKWrMdMMpoCJLC0myDwoUuGvk0dycTpm5NwUX6CnqDtYhtGkYg8
JT+LtG67k6yjXNa9wrE6VBJynDDPnlL8gLUl9ZCIFrmevJ75rOCaLsoFsghHPwIDAQABoyEwHzAd
BgNVHQ4EFgQULyCNnvW9Ngy5zM7Waf205mR11ZAwDQYJKoZIhvcNAQELBQADgYEApWlSy2GUSaHM
Kkc8XZmdQ0//SId8DKRKaFaZW388K4dJGTSWUnzq4iCWFrAN9O4D1DBnNE+dCDEmV8HvmyQBedsG
JnAre0VisqKz9CjIELcGUaEABKwkOgLe1YyqV29vS4Y3PuTxAhbkyphFb5PxjHDHH/WLQ8pOTbsC
vX4wO04=
-----END CERTIFICATE-----
```

Add IDP certificate entry to the .jks file

During SAML configuration, you must add the IDP certificate entry to the new `.jks` file you created as part of this sub-task "Create a keystore (*.jks) file" on page 99. Follow these steps to add the IDP certificate entry to the `.jks` file you created for that sub-task:

1. Navigate to where the server certificate exists. By default, it is located at:

`$EIS_HOME\repository\resources\security`

2. Extract X509 certificate Data for `Idp.xml`.

3. Copy `saml-idp.cer` to the `.jks` file by executing the following command:

```
C:\Program Files\Java\jdk1.7.0_67\jre\bin>keytool -import -trustcacerts -alias
mykey -file saml-idp.cer -keystore studentkeystore.jks
Enter keystore password: password
Owner: CN=WSO2 IDP, OU=Unknown, O=Unknown, L=Unknown, ST=Unknown, C=Unknown
Issuer: CN=WSO2 IDP, OU=Unknown, O=Unknown, L=Unknown, ST=Unknown, C=Unknown
```

```
Serial number: 12c20870
Valid from: Mon Aug 18 18:37:51 IST 2014 until: Tue Aug 18 18:37:51 IST 2015
Certificate fingerprints:
MD5:  8B:65:A6:A0:0F:F3:EA:B6:2A:32:37:7A:21:B5:CF:B6
SHA1: C5:73:6C:FD:63:15:45:C4:74:CF:E2:9D:DE:18:9A:4B:F9:6C:9C:5C
SHA256: 33:47:F1:95:1E:E7:DD:8B:F9:5C:17:A1:88:82:3E:0D:8B:B9:5C:9E:22:10:0B:57:
8F:51:62:9E:FF:1B:38
Signature algorithm name: SHA256withRSA
Version: 3
Extensions:
#1: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: 2F 20 8D 9E F5 BD 36 0C   B9 CC CE D6 69 FD B4 E6  / ....6.....i...
0010: 64 75 D5 90 du..
]
]
Trust this certificate? [no]:  yes
Certificate was added to keystore
```

4. To verify the certificate was added, execute the following commands:

```
C:\Program Files\Java\jdk1.7.0_67\jre\bin>keytool -list -v -keystore
studentkeystore.jks Enter keystore password: password
Keystore type: JKS
Keystore provider: SUN
Your keystore contains 2 entries
Alias name: mykey
Creation date: Apr 6, 2015
Entry type: trustedCertEntry
Owner: CN=WSO2 IDP, OU=Unknown, O=Unknown, L=Unknown, ST=Unknown, C=Unknown
Issuer: CN=WSO2 IDP, OU=Unknown, O=Unknown, L=Unknown, ST=Unknown, C=Unknown
Serial number: 12c20870
Valid from: Mon Aug 18 18:37:51 IST 2014 until: Tue Aug 18 18:37:51 IST 2015
Certificate fingerprints:
MD5:  8B:65:A6:A0:0F:F3:EA:B6:2A:32:37:7A:21:B5:CF:B6
SHA1: C5:73:6C:FD:63:15:45:C4:74:CF:E2:9D:DE:18:9A:4B:F9:6C:9C:5C
SHA256: 33:47:F1:95:1E:E7:DD:8B:F9:5C:17:A1:88:82:3E:0D:8B:B9:5C:9E:22:10:0B:57:
8F:51:62:9E:FF:1B:38
Signature algorithm name: SHA256withRSA
Version: 3
Extensions:
#1: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: 2F 20 8D 9E F5 BD 36 0C   B9 CC CE D6 69 FD B4 E6  / ....6.....i...
0010: 64 75 D5 90 du..
]
]
*******************************************
*******************************************
```

```
Alias name: mykey
Creation date: Apr 6, 2015
Entry type: PrivateKeyEntry
Certificate chain length: 1
Certificate[1]:
Owner: CN=John Doe, OU=Ellucian, O=Ellucian, L=Malvern, ST=PA, C=US
Issuer: CN=John Doe, OU=Ellucian, O=Ellucian, L=Malvern, ST=PA, C=US
Serial number: 126891cb
Valid from: Mon Apr 06 16:06:54 IST 2015 until: Thu Mar 31 16:06:54 IST 2016
Certificate fingerprints:
MD5:  D7:A6:90:A0:7D:19:DF:7E:D9:FF:01:5B:18:1D:FE:71
SHA1: 46:24:3E:A0:1E:65:76:21:D2:93:0F:29:76:60:17:40:07:0C:72:58
SHA256: ED:1B:C7:B5:07:49:80:4A:91:93:87:A1:15:9A:20:23:A7:BB:8B:99:89:02:47:
5F:5C:6E:42:47:AA:68:55
Signature algorithm name: SHA256withRSA
Version: 3
Extensions:
#1: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: 6F 8C FC 5C BA F1 11 FF   E2 58 C8 4F 2F 60 DA 2B  o..\.....X.O/`.+
0010: A2 EA D8 78                                        ...x
]
]
*******************************************
*******************************************
```