# webMethods Suite Security Infrastructure Guide

## LoginModules Guide

Version 9.8

April 2015

Security Infrastructure

**Document ID: SINEXT-USING-98-20150324**

## Table of Contents

# Preface

This document introduces the implementation of JAAS `LoginModules`. Developers of webMethods products will benefit most from the information in it. The information on the different types of default `LoginModules` will help the development of authentication solutions based on SIN.

This document describes the main functionality of SIN's components.

The topics for the required configuration are organized under the following headings:

**Preparing JAAS Configuration Files**
**Creating Technical User Credential Files**
**Creating Internal User Repository Files**

The topics for the authentication process and the usage of Security Infrastructure login modules are organized under the following headings:

| | |
|---|---|
| ● | **Authentication Process** |
| ● | **Configuration Parameters** |
| ● | **Predefined Login Modules** |
| ● | **Developing** `LoginModules` |
| ● | **Using the LDAP Framework** |

# I Preparing JAAS Configuration FilesCreating Technical User Credential FilesCreating Internal User Repository FilesCreating Custom Keys and Certificates

You use JAAS configuration files to manage authentication against multiple components and products of the webMethods Product Suite. JAAS configuration files allow you to define a uniform and flexible mechanism of authentication. They have commonly structured components which you can easily configure in order to authenticate successfully against multiple products, applications, or processes. The JAAS configuration comprises one or more than one login modules, which are grouped in a login context. The login modules define the actual authentication mechanism, and allow you to easily manipulate the overall authentication by configuring the behavior of a particular login module. A login context that is part of the JAAS configuration file controls and invokes the login modules in a pre-configured order. Every login context is a different login mechanism and it is up to the product to choose which one to use according to complete the use case.

Software AG Security Infrastructure comprises a set of pre-defined login modules which enable you to authenticate within the products of the webMethods Product Suite. Security Infrastructure login modules are reusable entities, which you can easily organize and configure in a uniform login context of a JAAS configuration file. Thus, you can define simply the rules of authentication of a particular business scenario, in the means of correct configuration of its JAAS file. The JAAS configuration files comprise the following components:

- Login Context
- Login Modules
- Classification of Login Modules
- Configuration Properties of Login Modules
- Comments within the JAAS file that describe the components

**Note:** When deploying JAAS configuration on the IBM WebSphere Application server fails, remove all comments from the configuration file.

The following tasks describe how you can organize and configure the components of a JAAS configuration file in a uniform way and authenticate against the products.

## Creating Login Contexts

A login context is a grouping of login modules in a JAAS configuration file. It provides the basic methods for user authentication. The stack of login modules allows you to configure applications or products to use more than one login module. The JAAS framework allows for a very flexible handling of stacks of login modules. When authenticating, the calling program instantiates directly the login modules that are grouped in the login contexts.

The sample excerpt below outlines a login context that contains the following predefined login modules that are provided by Security Infrastructure (`X509CertificateLoginModule`, `SSXLoginModule`, and `CentraSiteServerLoginModule`). The login modules are specified in the login context with their full class name (for example, `com.softwareag.security.jaas.login.modules.X509CertificateLoginModule`). The classification of the login modules is defined by flags (the flags used in the sample below are `required`, `requisite`, `optional`). The flags are specified after the login modules names. At the end of each login module definition are placed the parameters that control the behavior of the module. In the example below, the `X509CertificateLoginModule` has six parameters, while the other two modules have only one parameter respectively. All login modules are separated in the login context by semi-colons (;). Semi-colons separate the login contexts as well.

```
SoftwareAGSampleLoginContext
{
   com.softwareag.security.jaas.login.modules.X509CertificateLoginModule required
       check_crl_status=true
       crl_url="${com.softwareag.security.crl.url}"
       truststore_url="${com.softwareag.security.truststore.url}"
       truststore_password="${com.softwareag.security.truststore.password}"
       truststore_type=jks
       overwrite_username=false;

    com.softwareag.security.jaas.login.ssx.SSXLoginModule requisite
       template_section=OS;

   com.softwareag.security.jaas.login.xmlserver.CentraSiteServerLoginModule optional
       XMLSERVER_URL="http://localhost:53305/CentraSite/CentraSite";
};
```

To succeed the overall login process, the login modules have to succeed depending on the classification that is set to them.

## Defining Login Modules

The process of authentication includes the successful calling of a login module. Login modules can prompt for and verify a user name and a password, a client certificate, or enquire for user details from a user repository. The JAAS configuration specifies the login module that is to be used with a particular product or application. You can define a set of login modules within the JAAS configuration file. Moreover, you can configure the specific behavior of the login modules depending on the application requirements. You include the login modules in the login context using their full class name. The following samples outline the correct login modules entries.

```
com.softwareag.security.jaas.login.modules.X509CertificateLoginModule
com.softwareag.security.jaas.login.ssx.SSXLoginModule
com.softwareag.security.jaas.login.xmlserver.CentraSiteServerLoginModule
```

**To use the standard JAAS login modules with Software AG Runtime**

1. Open the *<SoftwareAG_directory>/profiles/CTP/configuration/config.ini* file.
2. Change the value of the `com.softwareag.platform.jaas.enabled` parameter from true to false.
3. Restart Software AG Runtime.

## Configuring the Classification of Login Modules

JAAS specification classifies the login modules depending on their status towards the successful authentication. Depending on the particular classification of the login module, you can configure it to take a significant role in the overall authentication process, or leave it as an optional element to the overall success. The following classifications of login modules are available:

- `Requisite`

  The login module is required to succeed. If it succeeds, the authentication proceeds down the login module list that is defined in the login context. If it fails, the control is immediately returned to the application and the authentication does not proceed down the login module list.

- `Required`

  The login module is required to succeed. If it succeeds or fails, the authentication process still proceeds down the login module list that is defined in the login context.

  **Note:** The overall authentication succeeds only if all `requisite` and `required` login modules succeed.

- `Sufficient`

  The login module is not required to succeed. If it succeeds, the control is immediately returned to the application and the authentication does not proceed down the login module list. If it fails, the authentication proceeds down the login module list.

  **Note:** If a sufficient login module is configured and succeeds, then the overall authentication succeeds only if the previous `requisite` and `required` login modules succeeded.

- `Optional`

  The login module is not required to succeed. If it succeeds or fails, the authentication process still proceeds down the login module list.

  **Note:** If there are not configured `requisite` or `required` login modules then the overall authentication succeeds only if at least one `sufficient` or `optional` login module succeeds.

## Configuring the Parameters of Login Modules

The behavior of a specific login module that is included into the context list depends on the parameters that are set to it and used during the authentication process. JAAS configuration files allow you to modify, in the means of functionality, the behavior of the used login modules. To configure a login module, you can list a set of parameters that are available for the particular login module, and provide values to them, which are essential to the authentication. You define the parameters of a login module in the login context, after the classification information. You can add more than one parameter and you separate the parameters using a space or a new line.

You can also add the `domain` parameter in your login modules. This parameter enables a dynamic use of login modules. To activate the domain usage, you must add the `domain` parameter to the *jaas.config* file for the particular login module. When the user logs in providing a domain and user name, the login modules in the jaas.config file verify the provided domain value and begin the authentication process for the user only if the provided domain value corresponds to the one defined for the specific login module. This behavior makes it possible for many consumers to share the same configuration by dynamically modifying the authentication logic in each use case.

**Note:** The domain usage is implemented for the `InternalLoginModule` and the `LDAPLoginModule`.

The full property list of the Security Infrastructure login modules that are provided by Software AG is available in the *Predefined Login Modules* section.

The JAAS configuration file now supports location tokens (@path and @url). For more information about path token support, see the *Working with Software AG Runtime* documentation.

## Specifying JAAS Configuration Files in Java Runtime

To use the created JAAS configuration file, you must point it to the installed Java Runtime Environment. You can specify the file in the JRE using the instructions below.

Specifying a JAAS configuration file must be done in the profile's config.ini file.

### To specify a JAAS configuration file in the *config.ini* file

1. Navigate to the config.ini file in the profile *<install-dir>/profiles/<profile>/config.ini*
2. Open the file with a text editor.
3. Change the value of the `java.security.auth.login.config` property to point to a valid JAAS configuration file location.

   For example,
   `java.security.auth.login.config=@url\:osgi.configuration.area/jaas.config`.
4. Save your changes and close the *config.ini* file.

**Note:** Only the content of the JAAS configuration file under: *<install-dir>/profiles/<profile>/configuration/jaas.config* will be migrated in future releases. You can still use a JAAS configuration file in a different location but you will have to migrate the file manually.

## Next Steps

If authentication is successful, JAAS creates a subject that contains one or more principals with security related attributes like passwords and cryptographic keys.

# 1 Creating Technical User Credential Files

Software AG Security Infrastructure provides a tool (*createTechUserCreds.exe* and *createTechUserCreds*) with which you can create technical user credential files. At a later stage, you use these files with the `SSXLoginModule` and thus search for and discover LDAP users securely on LDAP servers that do not support anonymous requests. By default, the tool is available in the following directory on the file system: *C:/`Software AG_directory`/common/security/ssx_32(64)/bin/*. To start the *createTechUserCreds* tool, you can use a command prompt. When you start the tool, you enter a user name and a password which are then encrypted and provided in the result text file.

Optionally, you can specify and use a key file to encrypt the technical user content in the result. A key file is an alternative file that is used for encryption of the result. The file encloses a string of 64 hexadecimal ASCII characters (digits 0-9, and lower case letters a-f). The initial 32 characters denote the alternate AES key and the final 32 characters denote the initialization vector.

**Note:** To use this tool, the SSX libraries must be in the library path of the system environment settings (the exact name of the property is different for the different operating systems). SSX libraries are located in the bin directory on Windows and in lib directory on UNIX based operating systems.

**To create a technical user credentials file**

1. Using the command prompt, open the following directory: `Software AG_directory`*\ common\runtime\security\bin*

   You cannot start the tool from a different location on the file system.

2. Depending on the operating system, start the tool using one of the following commands:

   - Windows

     ```
     createTechUserCreds.exe -f <result file name> -k <key file name> <user ID>
     ```

   - UNIX

```
./createTechUserCreds -f <result file name> -k <key file name> <user ID>
```

When you execute the tool without specifying an argument for the result file name, it still creates a text file with the corresponding technical user credentials. The file is created in the same directory in which you started the tool and has a predefined default name (*techuser*). To customize the invocation of the tool in the means of invocation parameters, you can use a set of pre-defined optional arguments. The available arguments and the corresponding descriptions are as follows:

| Argument | Description |
|---|---|
| `-f` | Provide a name for the result text file which contains the technical user credentials. If you do not use this argument the tool creates a default result file. |
| `-k` | Provide an alternative key file to encrypt the result text file that contains the technical user credentials. |
| `user ID` | Provide full DN of the technical user or user name. |

3 Press Enter and then provide the password.

**Example**

```
createTechUserCreds.exe -f <result file name> -k <keystore file name> <user ID>
```

```
./ createTechUserCreds -f <result file name> -k <keystore file name> <user ID>
```

The following examples provide information about more typical use cases of the tool:

```
createTechUserCreds.exe -f techUser.txt cn=testuser,dc=testdomain,dc=com
```

```
createTechUserCreds.exe -f techUser.txt -k key.keystore
cn=testuser,dc=testdomain,dc=com
```

The tool creates a text file, which contains the encrypted technical user credentials, and stores it in the same directory in which you started it. As a next step, you can provide the file to the `SSXLoginModule` and search for LDAP users.

# 2 Creating Internal User Repository Files

You can create and/or modify internal user repository files that contain user names and their respective encrypted passwords. Currently, there are two Software AG Security Infrastructure tools that you can use for this purpose: Internal User Repository Command Line Tool and the ssxtxt-passwd tool. Software AG recommends the usage of Internal User Repository Command Line Tool.

The information is organized under the following headings:

## Internal User Repository Command Line Tool

The start scripts of the tool, *internaluserrepo.bat* and *internaluserrepo.sh*, are in the *<SoftwareAG_directory>/common/bin* directory. At a later stage, you can use the user repositories files with login modules that have a property for using such files (for now, these login modules are `InternalLoginModule` and `SSXLoginModule`).

### To create and/or modify an internal user repository file

1 Use the command prompt to open the *<SoftwareAG_directory>/common/bin* directory.

2 Depending on the operating system, start the tool using one of the following commands:

- Windows

```
internaluserrepo.bat [-f <filename>] [-c] [-p <password>] [-d | -e] <userId>
```

- UNIX

```
./internaluserrepo.sh [-f <filename>] [-c] [-p <password>] [-d | -e] <userId>
```

where

| Argument | Description |
|---|---|
| `-h`, `-help` | Prints guidelines for using the tool. |
| `-f`, `-file` | Specifies the user repository file. |
| `-c`, `-create` | Creates a text repository file. You can specify the location and file name with the `-f` argument followed by the wanted URL (path and name of the file to be created). If only the `-c` option is specified, a file, named *users.txt*, is created in the execution directory of the tool. If you do not use the `-c` argument and the specified text file does not exist, an error is returned. If you specify `-c` and the file already exists, the new information is added at the end of the repository file. |
| `-p`, `-password` | Provides the specified password on the command line.<br><br>**Note:** Passwords can contain only digits, Latin letters, and the following characters: ! ( ) - . ? [ ] _ ~. They cannot exceed 128 characters. |

| Argument | Description |
|---|---|
| `-d`, `-delete` | Deletes the credentials for the specified user from the text repository file. If you do not specify a file and a *users.txt* file exists in the directory of the tool, the user is removed from this file. |
| `-e`, `-existing` | Checks whether the specified user exists in the text repository file. You should provide a URL to the file using the `-f` argument. |
| *`<userId>`* | Contains the user name for the text repository file operation. If you call the tool only with this option and a *users.txt* file exists in the directory of the tool, a new user with a user name *`<userId>`* is added in the file. Then, a prompt asks for the user password. If a user with this userId already exists in the repository file, the password is changed. **Note:** User names can contain only digits, Latin letters, and the following characters: ! ( ) - . ? [ ] _ ~. They cannot exceed 128 characters. |

**Note:** The only required parameter is `userId`.

### Status Codes

Internal User Repository Command Line Tool returns exit codes that define the result of the execution. If the command is executed successfully, no exit status is returned.

You can see the descriptions of the exit codes in the following table:

| Exit code | Description |
|---|---|
| -1 | The specific `userId` that is searched for (option `-e`) does not exist in the repository file. |
| 1 | The password is not set. Please specify a password. |
| 2 | The `userId` is too long. The maximal length for a `userId` is 128 characters. |
| 3 | The `userId` contains an invalid character. |
| 4 | The password contains an invalid character. |
| 5 | The password is too long. The maximal length for a password is 128 characters. |
| 6 | The repository file is inconsistent. Multiple version occur in the repository file. |
| 7 | The repository file is inconsistent. The version is invalid. |
| 8 | The repository file is inconsistent. The repository version is not specified. |
| 9 | The repository file cannot be opened or created. |
| 10 | The `userId` is missing. |
| 11 | The specified parameter is conflicting or invalid. |

## ssxtxtpasswd Tool

Software AG Security Infrastructure provides also another tool (*ssxtxtpasswd.exe*, *ssxtxtpasswd*) with which you can create internal user repository files. At a later stage, you use these files with the `SSXLoginModule`. By default, the tool is available in the following directory on the file system: *`Software AG_directory`\ common\runtime\security\bin*. To start the *ssxtxtpasswd* tool, you use a command prompt. When you start the tool, you enter a user name and a password which are then encrypted (SHA512 and Base64) and provided in the result text file. The tool adds new or replaces existing user credentials in the text file.

**Note:** When you enter a user name, you can use only digits, Latin letters, and the following characters: ! ( ) - . ? [ ] _ ~ . When you enter a password, you can use only digits, Latin letters, and the following characters: !"#$%&'()*+,-./:;<=>?[\]^_`{|}~.

**To create and/or modify an internal user repository file**

1 Using the command prompt, open the following directory:

   *`Software AG_directory`\ common\runtime\security\bin*

   You cannot start the tool from a different location on the file system.

2 Depending on the operating system, start the tool using one of the following commands:

   - Windows

     ```
     ssxtxtpasswd.exe [-c] [-f <result file name>] [-p <password>] [-d] <user ID>
     ```

   - UNIX

     ```
     ./ssxtxtpasswd [-c] [-f <result file name>] [-p <password>] [-d] <user ID>
     ```

   To customize the invocation of the tool in the means of invocation parameters, you can use a set of pre-defined optional arguments. The available arguments and the respective descriptions are as follows:

| Argument | Description |
|---|---|
| `-f` | Provide a name for the result text file which contains the user credentials. If you do not use this argument the tool creates a default result file called *ssx_user*. |
| `-c` | Using this parameter, you create a text repository file with a specified name (`-f` parameter). If you do not use the `-c` parameter and the specified text file does not exist, an error is returned. If you specify `-c` and the file already exists, `-c` argument is ignored and the tool does not create a new file. When you execute the tool without specifying an argument for the result file name (`-f` argument), it still creates a text file with the corresponding internal |

| Argument | Description |
|---|---|
| | user repository information. The file is created in the same folder in which you started the tool and has a predefined default name (*ssx_user*). |
| `-p` | Provide a password directly on the command line. Thus, the tool does not invoke a non-echo input of the password in the next steps. |
| `-d` | Remove credentials data for a particular user from the text repository file. When you use the `-d` parameter, the tool ignores the presence of the `-c` parameter. |
| `user ID` | Provide user name which you want to add or replace in the text file. |

3 Press Enter and then provide the password.

**Example**

The following examples provide information about more typical use cases of the tool:

```
ssxtxtpasswd.exe -c -f internalUser.txt -p pass myUser
```

```
ssxtxtpasswd.exe -c -f internalUser.txt -p newpass myUser
```

```
ssxtxtpasswd.exe -d -f internalUser.txt myUser
```

The tool creates a text file, which contains the encrypted internal user repository credentials, and stores it in the same directory in which you started it. As a next step, you can provide the file to the `SSXLoginModule` and search for INTERNAL users.

# 3 Creating Custom Keys and Certificates

Software AG Common Platform provides a single sign-on service which has predefined keystore (*keystore.jks*) and truststore (*platform_truststore.jks*). The predefined keystore and truststore contain default keys used for issuing and validating signed SAML assertions. You can create and modify these keystore and certificates using the *certtool* tool provided by Software AG Security Infrastructure.

The *certtool* tool is located in the `Software AG_directory`\*common*\*bin* folder. It is a wrapper of Java keytool and has default options that are used if the user does not provide any custom input.

**Notes:**

1. After you create a new certificate and add it to the keystore, you must also update the configuration of the SSOS service for your changes to take effect.
2. If the keystore file already exists, and you try to generate a new key pair in the same keystore file, a warning is displayed, stating that the file will be overwritten.

**To use the *certtool* tool**

1 Using the command prompt, open the following directory: `Software AG_directory`\\ *common*\*bin*. You cannot start the tool from a different location on the file system. Depending on the operating system, start the tool using one of the following files:

- Windows

  ```
  certtool.bat
  ```

- UNIX

  ```
  ./certtool.sh
  ```

2 To generate a key pair, type the following command:

```
certtool.bat/sh -generate
```

You are prompted to provide a common name (CN) for the certificate.

The keystore certificate is generated in the location specified by the `DEFAULT_PATH` option.

3 To add the newly generated .cer file to the truststore, type the following command:

```
certtool.bat/sh -add
```

Follow the prompts. The .cer file is added to the location specified by the `TRUSTSTORE_FILE` option.

4 To list the keystore contents, type the following command:

```
certtool.bat/sh -listkeystore
```

Follow the prompts. The keystore contents are listed in the command prompt.

5 To list the truststore contents, type the following command:

```
certtool.bat/sh -listtruststore
```

Follow the prompts. The truststore contents are listed in the command prompt.

6 To delete a certificate from the truststore, type the following command:

```
certtool.bat/sh -delete
```

You are prompted to provide the alias name of the certificate file to be deleted.

### Available Commands

Below is the list of commands available in the *certtool.bat/sh* file:

| Argument | Description |
|---|---|
| `-listkeystore` | Lists the keystore certificates currently located in the keystore. The default keystore certificate is default.jks with a default password manage.<br><br>**Note:** The keystore should contain only one keystore certificate which is used for issuing signed SAML assertions. |
| `-listtruststore` | Lists the truststore certificates currently located in the truststore. The default certificate is default_truststore.jks with a default password manage.<br><br>**Note:** The truststore can contain multiple public truststore certificates which are used for validating SAML assertion signatures. |
| `-add` | Adds a trusted certificate to the truststore. The default_truststore.jks certificate is used if no other certificate is specified. |
| `-delete` | Deletes a trusted certificate from the truststore. |
| `-generate` | Generates a key pair and exports the public information as a .cer file. |

| Argument | Description |
|---|---|
| `-usage` | Prints the available commands. |

### Available Options

Below is a list of options available in the *certtool.bat/sh* file.

**Caution:** All options in the table below have default values assigned to them. Please note that you are advised to modify them with extreme caution.

| Option | Description |
|---|---|
| `DEFAULT_PATH` | Default path where the certificate stores will be created, for example C:\Software AG\common\conf.<br><br>The value is automatically provided when you install the *certtool* using the Software AG Installer. |
| `KEYTOOL_PATH` | Default path to the Software AG Java keytool, for example C:\Software AG\jvm\jvm170_32\bin\keytool.<br><br>The value is automatically provided when you install the *certtool* using the Software AG Installer. |
| `KEYSTORE_KEY_ALIAS` | Alias keystore name.<br><br>Default value is default. This value will be used if no other alias is specified. |
| `KEYSTORE_FILE` | Value for the name and location of the created keystore certificate.<br><br>If no other value is specified, the *certtool* generates a keystore certificate with the name "default.jks" in C:\Software AG\common\conf. |
| `KEYSTORE_TYPE` | The type of the keystore.<br><br>Default value is JKS. |
| `KEYSTORE_PASSWORD` | The password for the keystore. The default value is manage. |
| `TRUSTED_CERT_ALIAS` | Alias truststore certificate name.<br><br>Default value is default. This value will be used if no other alias is specified. |
| `TRUSTSTORE_FILE` | Value for the name and location of the created truststore.<br><br>If no other value is specified, the *certtool* generates a keystore certificate with the name "default_truststore.jks" in C:\Software AG\common\conf. |
| `TRUSTSTORE_TYPE` | The type of the truststore.<br><br>Default value is JKS. |
| `TRUSTSTORE_PASSWORD` | The password for the truststore.<br><br>Default value is manage. |
| `X509_FILE` | Value for the name and location of the created truststore certificate. |

| Option | Description |
|---|---|
| | If no other value is specified, the *certtool* generates a certificate with the name "default.cer" in C:\Software AG\common\conf. |
| `VALIDITY` | The validity of the certificate in days.<br><br>Default value is 1826. |
| `KEY_ALGORITHM` | Specifies the algorithm to be used to generate the key pair.<br><br>Default value is RSA. |
| `SIG_ALGORITHM` | Specifies the algorithm that should be used to sign the self-signed certificate. This algorithm must be compatible with `KEY_ALGORITHM`. Its value is derived from the algorithm of the underlying private key.<br><br>For example, if the private key is of type `DSA`, the value of the `SIG_ALGORITHM` option is SHA1withDSA. |
| `KEY_SIZE` | Specifies the size of each key to be generated.<br><br>Default value is 1024. |

# II Authentication ProcessConfiguration ParametersPredefined Login ModulesDeveloping Login ModulesUsing the LDAP Framework

This chapter describes how the Software AG Security Infrastructure operates. The information is useful for developers who want to implement the `LoginModules`.

The information is organized under the following headings:

## Overview

Following is an overview of the authentication process in SIN:

1. An application instantiates a `LoginContext`
2. The `LoginContext` consults a Configuration to load all of the `LoginModules` configured for that application name.
3. The application invokes the `LoginContext`'s login method
4. The login method invokes all of the loaded `LoginModules`

   Each `LoginModule` attempts to authenticate the subject. Upon success, `LoginModules` associate relevant `Principals` and `credentials` with a `Subject` object that represents the subject being authenticated.
5. The `LoginContext` returns the authentication status to the application
6. If authentication is successful, the application retrieves the `Subject` from the `LoginContext`, otherwise the `LoginException` will be thrown

## Authentication Steps

### To authenticate a user in SIN

1 Define the *jaas.config* file.

Each `LoginModule` has specific parameters that must be defined in the *jaas.config* file.

2 Define the properties file for log4j.

Following is an example of a properties file for log4j:

```
# Set root logger level to INFO and its only appender to A1.
log4j.rootLogger=INFO, A1

# A1 is set to be a ConsoleAppender.
log4j.appender.A1=org.apache.log4j.ConsoleAppender

# A1 uses PatternLayout.
log4j.appender.A1.layout=org.apache.log4j.PatternLayout
log4j.appender.A1.layout.ConversionPattern=%d{ABSOLUTE} [%t] %-5p %c %x - %m%n
```

See *Troubleshooting* for additional information on how to handle log4j.

3 Develop the JAAS client.

4 Load the JAAS configuration.

There is one configuration available per JVM. This configuration can contain one or many application contexts, which in turn consist of one or many `LoginModules`. The JAAS configuration file can be loaded in two different ways depending on the environment:

- Inside the Common Runtime, the JAAS configuration file will be loaded from location that is pointed by the `java.security.auth.login.config` property in the profile's config.ini file. The location of the jaas.config file is specified in the value of the `java.security.auth.login.config` property.
- Outside of the Common Runtime, the JAAS configuration will be loaded from location that is pointed by `java.security.auth.login.config` Java system property. For this approach, a variable has to be set by the application either at start time as a parameter of a Java VM or programmatically.

  Set the variable as a Java VM system property:

```
-Djava.security.auth.login.config=<URL to configuration>
```

5 Set up the credentials.

Software AG Security JAAS Stack provides the `SagCredentials` class. All `LoginModules` support only this type of credentials.

`SagCredentials` are queried by `SagCallbackHandler`, which is the default callback handler for credentials. It supports `SagCredentialCallback`.

Upon successful authentication, the `SagCredential`s can be stored as private credentials in the Subject, from where they can be retrieved by the application.

Following is a list of user's attributes that `SagCredentials` sets and retrieves:

- `Domain name`
- `Password`
- `User name`
- `X.509 certificate chain` including `user certificate` and the `issuer certificate` (excluding the `root certificate`)
- SAML artifact

6 Create the `LoginContext`.

Following is an example of how to authenticate a user. In this case, you must instantiate a `LoginContext`:

```
import javax.security.auth.login.LoginContext;
. . .
LoginContext loginContext =
    new LoginContext(<configuration_entry_name>,
           <CallbackHandler_to_be_used_for_user_interaction>);
```

< configuration entry name > is the name used as the index into the *jaas.config* file.

7 After the user is authenticated, the `Subject` is derived from the `LoginContext`.

8 Different types of `Principles` are derived from an available `Subject`.

The `Principals` architecture in SIN is based on an abstract class - `AbstractSagPrincipal` - and all other SAG `Principals` extend it. SIN provides some implemented classes for common use cases: `SagUserPrincipal`, `SagGroupPrincipal`, `SagRolePrincipal`, `LightWeightPrincipal`. SIN returns no or only one user principal for the authenticated user. It is configurable in the JAAS configuration.

# 4 Configuration Parameters

These configuration parameters are the parameters in the `SagAbstractLoginModule` and are global for all `LoginModules`.

They are referred to as "global parameters" because they apply to all types of `LoginModules`.

`LoginModules` are configured in two ways:

- By means of `Flags` that influence the whole process of authentication
- By means of the module-specific parameters that are customized for the module and influence the behavior of the module only

For details on the `Flags`, see *Preparing JAAS Configuration Files*

For a list of the module-specific parameters, see the description of the respective login module.

The following table outlines the global configuration parameters.

| Parameter | Description |
|---|---|
| `create_user_principal` | Optional.<br><br>You use it to define whether the `commit ()` method creates a `SagUserPrincipal` using the `SagCredentials` available in the `sharedState Map`.<br><br>Valid values are:<br><br>■ true - The `commit ()` method creates a `SagUserPrincipal`.<br><br>■ false - Default value. The `commit ()` method does not create a `SagUserPrincipal`.<br><br>The login modules that do not create `SagUserPrincipal` in their own `commit ()` method must call the `super.commit ()` method. |

| Parameter | Description |
|---|---|
| | **Note:** The SagUserPrincipal is created only once. Once set to "true", this flag cannot be changed. |
| store_credentials | Optional.<br><br>You use it to define whether to store SagCredentials in Subject.privateCredentials. The servlet context and header field of SagCredentials are not stored.<br><br>Valid values are:<br><br>■ true - Default value. SagCredentials is stored in Subject.privateCredentials.<br>■ false - SagCredentials is not stored in Subject.privateCredentials. |
| keep_password | Optional.<br><br>You use it to define whether to keep the password (if present in SagCredentials) in the credentials that are stored in Subject.privateCredentials.<br><br>Valid values are:<br><br>■ true - Default value. The password (if present in the SagCredentials) is kept in the credentials that are stored in the Subject.privateCredentials.<br>■ false - The password (if present in the SagCredentials) is not kept in the credentials that are stored in the Subject.privateCredentials.<br><br>**Note:** This parameter requires the store_credentials parameter to be set to "true". |

**Important:** See *Predefined Login Modules* for guidelines on the module-specific parameters.

# 5 Predefined Login Modules

Security Infrastructure (SIN) provides default login modules.

You can use the default modules to do the following:

- Authentication, role, user, and group management
- Support additional backend authentication systems
- Exchange credentials through the `SharedStateMap` for internal SSO

The default modules are as follows:

For more information about deprecated login modules in Security Infrastructure, see *Deprecated Login Modules*

## SagAbstractLoginModule

`SagAbstractLoginModule` is the basic login module in SIN. It provides you with a `commit()` method that uses the global configuration parameters. See ***Configuration Parameters*** for details.

Use this login module for the following:

- Extend it to create your own login modules.
- Create the `SagUserPrincipals` with the information stored in the shared map through the authentication process.

When setting up the JAAS configuration, keep in mind the following basics:

- The SIN-based login contexts return zero or only one `SagUserPrincipal` if the authentication succeeds. When setting up the JAAS configuration, keep in mind that some applications expect only one `SagUserPrincipal` as the result of a successful authentication. If your application expects more than one user principal, you must configure the login context accordingly.
- Keeping the password in clear text in the `Subject.privateCredentials` may constitute a security risk, depending on how the Subject is handled. However, there are use cases where the password needs to be accessible through the Subject, so you must store the password only if needed.

## InternalLoginModule

### Description

You use the `InternalLoginModule` to authenticate against a user repository defined as a file on the file system. This is the default authentication mechanism for all webMethods suite products.

In case of successful authentication, the `InternalLoginModule` provides a user repository manager. It also creates a `SagUserPrincipal` object, and, optionally, a set of `SagGroupPrincipal` objects.

### Parameters

The following table outlines the configuration parameters for the `InternalLoginModule`.

| Parameter | Description |
|---|---|
| `domain` | Optional.<br><br>String. Specifies the domain name to be used for authentication. Applicable if the domain usage is activated for the `InternalLoginModule`. |
| `internalRepository` | Specifies the path to the internal user repository file. |
| `groupRepositoryPath` | Optional.<br><br>Specifies the path to the internal group repository file. |
| `create_group_principal` | Optional.<br><br>The parameter is meaningful only if groupRepositoryPath is specified.<br><br>Valid values are:<br><br>■ false - Default. The login module does not create group principals<br>■ true. The login module creates group principals based on the information contained in groupRepositoryPath and attaches the to the subject. |

### Repository Files Format

The user-defined repository files must comply with the following format:

```
*
* Default test repository for INTERNAL based authentication
*
* Copyright (c) 2001 - 2013 Software AG, Darmstadt, Germany and/or Software AG USA,
* Inc., Reston, VA, United States of America, and/or their licensors. All rights ↩
reserved.
version:3.0
*
*
user:username:$6a$kMpE+PvDv83zjcQe6fk7rWEiK80V73qoy90Zzr0J4p4W3K1g9x1w2zEadkEjL20Lm1cozDfKJD7ZJckE3AysKw==
*
```

The group repository files must comply with the following format:

```
*
*
* Default test repository for INTERNAL based authentication
*
* Copyright (c) 2001 - 2013 Software AG, Darmstadt, Germany and/or Software AG USA,
* Inc., Reston, VA, United States of America, and/or their licensors. All rights ↩
reserved.
version:3.0
*
*
admin:1:administrator,user2
testadmin:2:user2
*
```

### Example

The following sample excerpt outlines the INTERNAL mode of the `InternalLoginModule` and the corresponding configuration included in a login context of a JAAS configuration file.

```
LoginINTERNAL {
   com.softwareag.security.jaas.login.internal.InternalLoginModule required
         domain=
         logCallback=true
         create_group_principal=true
         internalRepository="/tmp/myrepo/internalUserRepo"
         groupRepositoryPath="/tmp/myrepo/internalGroupRepo";
};
```

## LDAPLoginModule

### Overview

The `LDAPLoginModule` enables you to use an external directory to authenticate users. For more information about using internally defined users and groups, see ***InternalLoginModule***.

You can define your JAAS configuration to access information from an external directory if your site uses one of the following external directories for user and group information:

- Lightweight Directory Access Protocol (LDAP)
- Active Directory acting as an LDAP server

### JAAS Configuration Properties

The following table outlines the JAAS configuration properties for all LDAP connections.

| Parameter | Description |
|---|---|
| `alias` | Optional.<br><br>Specifies the alias of the LDAP configuration entry. If not specified, it is set to match the `url` parameter.<br><br>A valid value is any string of characters. |
| `url` | Required.<br><br>Specifies the URL to the LDAP server. If you want to use an SSL connection to the LDAP server, the URL should start with ldaps, and you should provide truststore and/or keystore parameters. The expected format is: ldap://<host>:<port> or ldaps://<host>:<port>.<br><br>If `url` points to IPv6 IP (not domain name), it must be enclosed in square brackets. For example: alias=ldap://[::1]:389 |
| `domain` | Optional.<br><br>String. Specifies the domain name to be used for authentication. Applicable if the domain concept is activated for the `LDAPLoginModule`. |
| `noPrinIsAnonymous` | Optional.<br><br>When `prin` is not defined, this property specifies what credentials are used for LDAP server authentication.<br><br>Valid values are:<br><br>■ true - Default value. The connection to the LDAP server is done anonymously. |

| Parameter | Description |
|---|---|
| | ■ false - The real user credentials of the user that connects to the LDAP server are also used for LDAP authentication. |
| `prin` | Required only if `noPrinIsAnonymous` is set to false. Otherwise, this parameter must not be specified.<br><br>Specifies the distinguished name (DN) of the technical user that connects to the LDAP server if an anonymous access to the LDAP server is not allowed. |
| `cred` | Required only if `noPrinIsAnonymous` is set to false. Otherwise, this parameter must not be specified.<br><br>Specifies the password of the technical user that connects to the LDAP server. You use it together with the `prin` parameter.<br><br>A valid value is any string of characters. |
| `credHandle` | Can be used instead of `cred`. Handles passman storage for technical user passwords. When a login is successful, cred is placed in passman. |
| `timeout` | Specifies the maximum time in milliseconds to spend for an LDAP operation.<br><br>The default value is 5000. |
| `useaf` | Optional. Boolean.<br><br>Specifies whether to use affixes (dnprefix and dnsuffix) or not. Use the affixes for an easier construction of user DNs with less errors.<br><br>Valid values are:<br><br>■ true - The login module uses affixes.<br>■ false - Default value. The login module does not use affixes. |
| `dnprefix` | Optional. String.<br><br>Specifies the prefix to attach in front of the username when performing operations on the LDAP server. To use this parameter, you should have `useaf` set to true.<br><br>A valid value is any string of characters. |
| `dnsuffix` | Optional. String.<br><br>Specifies the suffix to attach after the username when performing operations on the LDAP server.<br><br>A valid value is any string of characters. |
| `usecaching` | Optional. Boolean.<br><br>Specifies if the LDAP framework caches users and/or groups.<br><br>Valid values are: |

| Parameter | Description |
|---|---|
| | ■ true - Default value. The LDAP framework caches all users and/or groups.<br>■ false - The LDAP framework does not cache any users and/or groups. |
| `poolmin` | Specifies the minimum number of objects to be kept in the cache. |
| `poolmax` | Specifies the maximum number of objects to be kept in the cache. |
| `mattr` | Optional.<br><br>The login module uses this parameter when performing member-search operations. The meaning of this parameter depends on the value of `memberinfoingroups`. If `memberinfoingroups` is true, the `mattr` parameter points from a group to the users that are members of this group. If `memberinfoingroups` is false, the `mattr` parameter points from a user entry to the groups that the user is a member of.<br><br>A valid value is any string of characters.<br><br>The default value is memberOf. |
| `memberinfoingroups` | Optional. Boolean.<br><br>Specifies whether the login module searches users in a group or groups in a user. You can use it only if the `mattr` parameter is applied to users or groups.<br><br>Valid values are:<br><br>■ true - The login module searches users in a group.<br>■ false - Default value. The login module searches groups in a user. |
| `createGroups` | Optional. Boolean.<br><br>Specifies whether to extract the groups of the logged-in user from the LDAP server.<br><br>Valid values are:<br><br>■ true - Default value. The login module extracts the groups of the logged-in user from the LDAP server.<br>■ false - The login module does not extract the groups of the logged-in user from the LDAP server. |
| `createUserProperties` | Specifies whether user properties should be populated to SagUserPrincipal.<br><br>Valid values are:<br><br>■ true - The user properties are populated to SagUserPrincipal<br>■ false - Default value. The user properties are not populated to SagUserPrincipal. |

| Parameter | Description |
|---|---|
| `createGroupProperties` | Specifies whether group properties should be populated to SagGroupPrincipal.<br><br>Valid values are:<br><br>■ true - The group properties are populated to SagGroupPrincipal<br>■ false - Default value. The group properties are not populated to SagGroupPrincipal. |
| `uidprop` | Optional<br><br>Specifies the LDAP username attribute.<br><br>The default value is CN. |
| `gidprop` | Optional.<br><br>Specifies the LDAP group attribute.<br><br>A valid value is any string of characters.<br><br>The default value is CN. |
| `userrootdn` | Optional.<br><br>Specifies the location to be searched for users.<br><br>A valid value is any string of characters. |
| `grouprootdn` | Optional.<br><br>Specifies from where to start searches for groups.<br><br>A valid value is any string of characters. |
| `groupobjclass` | Optional.<br><br>Specifies that the found object is a group. The login module uses this parameter when searching for groups.<br><br>The default value is group. |
| `personobjclass` | Optional.<br><br>Specifies that the found object is a person. The login module uses this parameter when searching for users.<br><br>The default value is person. |
| `truststoreUrl` | Specifies the URL of the truststore to be used if an SSL connection is required. |
| `truststorePassword` | Specifies the password for the used truststore if an SSL connection is required. |
| `truststoreType` | Specifies the type of truststore to be used if an SSL connection is required. |
| `keystoreUrl` | Specifies the URL of the keystore to be used if an SSL connection is required. |
| `keystorePassword` | Specifies the password for the used keystore if an SSL connection is required. |

| Parameter | Description |
|---|---|
| `keystoreType` | Specifies the type of keystore to be used if an SSL connection is required. |

### Example

The following sample excerpt outlines  and the corresponding configuration included in a login context of a JAAS configuration file.

```
ExampleRealm {

  com.softwareag.security.sin.is.ldap.lm.LDAPLoginModule sufficient
    alias="name1";

  com.softwareag.security.sin.is.ldap.lm.LDAPLoginModule sufficient
    alias="name2";

  com.softwareag.security.sin.is.ldap.lm.LDAPLoginModule sufficient;

  com.softwareag.security.sin.is.ldap.lm.LDAPLoginModule required
    alias="name3"
    url="ldap://localhost:389"
    prin="CN=sectest,OU=user,dc=example,dc=org"
    cred="******"
    useaf="true"
    dnprefix="CN="
    dnsuffix=",OU=user,dc=example,dc=org"
    usecaching="false"
    mattr="roleoccupant"
    memberinfoingroups=false
    creategroups=true
    gidprop="CN"
    grouprootdn="OU=Groups,dc=example,dc=org"
    groupobjclass="organizationalRole"
    personobjclass="organizationalPerson";
};
```

## SAMLAssertValidatorLoginModule

- Example

### Description

You use `SAMLAssertValidatorLoginModule` to validate the delegation ticket issued from `SAMLAssertIssuerLoginModule`. You can use it for both SAML 1.1 and SAML 2 assertion validation.

### Parameters

None.

### Example

The following sample excerpt outlines `SAMLAssertValidatorLoginModule` and the corresponding configuration included in a login context of a JAAS configuration file.

The following login context is installed by default with Software AG Common Platform.

```
/** Login context used in Common Platform for a default authentication **/
Default {
    // SSOS login module for SAML signed assertion validation
    com.softwareag.security.idp.saml.lm.SAMLAssertValidatorLoginModule sufficient;

    // Internal repository login module (java based)
    com.softwareag.security.jaas.login.internal.InternalLoginModule required
        template_section=INTERNAL
        logCallback=true
        internalRepository="C:/softwareag/common/conf/users.txt"
        create_group_principal=true
        groupRepositoryPath="C:/softwareag/common/conf/groups.txt";
};
```

## SAMLAssertIssuerLoginModule

- Description
- Parameters

- Example

## Description

You use `SAMLAssertIssuerLoginModule` to issue a SAML1.1 or SAML 2 assertion as a delegation ticket between Software AG products.

You can only use the `SAMLAssertIssuerLoginModule` in a chain of login modules. Using this login module on its own, in a separate login context, is not possible, because it is the other modules in a given login context that perform the actual authentication of the user. When the authentication is successful, `SAMLAssertIssuerLoginModule` issues a SAML assertion where the fully qualified name of the authenticated user is part of the `Subject` of the `AuthenticationStatement` attribute of the SAML 1.1 assertion and the `SubjectConfirmation` attribute of the SAML 2 assertion. Optionally, the assertion contains a list of groups (where such are available) as part of the `AttributeStatement` attribute of the SAML assertion.

## Parameters

The `SAMLAssertIssuerLoginModule` has a single parameter that you set in the JAAS configuration.

| Parameter | Description |
|---|---|
| `forceSamlVersion` | Optional.<br><br>Defines which SAML assertion version to use to issue the delegation token.<br><br>Valid values are:<br><br>■ `1.1` - Use this value to force SAML 1.1 assertion.<br>■ `2.0` - Default value. Use this value to force SAML 2 assertion. |

## Example

The following sample excerpt outlines `SAMLAssertIssuerLoginModule` and the corresponding configuration included in a login context of a JAAS configuration file.

First, `InternalLoginModule` authenticates the user. If the authentication is successful, `SAMLAssertIssuerLoginModule` issues a SAML 1.1 assertion to be used as a delegation ticket.

```
/** Login Configuration for the SAMLAssertIssuerLoginModule. **/
SAMLIssuerRealm {
    // Internal repository login module (java based)
    com.softwareag.security.jaas.login.internal.InternalLoginModule required
        template_section=INTERNAL
        logCallback=true
        internalRepository="C:/softwareag/common/conf/users.txt"
        create_group_principal=true
        groupRepositoryPath="C:/softwareag/common/conf/groups.txt";

    // SSOS login module for SAML 1.1 signed assertion issuance
    com.softwareag.security.idp.saml.lm.SAMLAssertIssuerLoginModule sufficient
        forceSamlVersion="1.1";
};
```

## JMXDelegatedAuthLoginModule

### Description

You use `JMXDelegatedAuthLoginModule` to validate the delegation ticket issued from `SAMLAssertIssuerLoginModule` or directly from the SSO service. You can use it for both SAML 1.1 and SAML 2 assertion validation. The purpose of this login module is to support the JMX delegation mechanism. The login module gets a delegation ticket from the password field of the supplied credentials.

### Parameters

None.

### Example

The following sample excerpt outlines `JMXDelegatedAuthLoginModule` and the corresponding configuration included in a login context of a JAAS configuration file.

The following login context is installed by default with Software AG Common Platform.

```
/*
 * Login context, used in Common Platform for management channel.
 */
PlatformManagement {
    // SSOS login module for SAML signed assertion validation
    // used for delegated authentication only for JMX
    com.softwareag.security.idp.saml.lm.JMXDelegatedAuthLoginModule sufficient;

    // Internal repository login module (java based)
    com.softwareag.security.jaas.login.internal.InternalLoginModule required
        template_section=INTERNAL
        logCallback=true
        internalRepository="C:/softwareag/conf/users.txt";
};
```

## ServletHeaderLoginModule

- Description
- Parameters
- Example

### Description

You use `ServletHeaderLoginModule` to extract information from an `HttpServletRequest` which is sent to the login module as part of the `SagCredentials`. The login module extracts the X.509 certificate chain or SAML artifacts, which are received as a result of an HTTPS with `ClientAuthentication` against a web server. The login module enters this information into the `SagCredentials` and makes it available to other login modules used in the login context of a JAAS configuration file. Optionally, the login module can extract more information, such as user names and passwords.

### Parameters

The following table outlines the parameters of `ServletHeaderLoginModule`.

| Parameter | Description |
|---|---|
| `saml_artifact_prop_name` | Optional.<br><br>Defines the name of the SAML artifact property. The default value is `SAMLArt`. |
| `netegrity_siteminder_prop_name` | Optional.<br><br>Defines the name of the Netegrity SiteMinder property. The default value is `SM_USER`. |

### Example

The following sample excerpt outlines `ServletHeaderLoginModule` and the corresponding configuration which is included in a login context of a JAAS configuration file.

```
/** Login Configuration for the ServletHeaderLoginModule. **/
ServletHeaderLogin {
    com.softwareag.security.jaas.login.modules.ServletHeaderLoginModule optional;
};
```

## SimpleNameMappingLoginModule

### Description

You use `SimpleNameMappingLoginModule` to map a user name that is in the `sharedState` or `CallbackHandler` to another user name, which is for example in a different user repository. The login module sends the result in the `sharedState` map. Depending on the parameters you include in the JAAS configuration file, you can provide different mapping modes with the login module. The properties mapping mode is based on a Java properties file. The regular expression mapping mode is based on the *java.util.regex* package. To enable a mapping mode you must use the corresponding configuration parameter in the JAAS configuration. Note that you cannot use both mapping modes at the same time.

For more sophisticated mapping method, you can sub-class `SimpleNameMappingLoginModule`. Using the following sample excerpt, you can rework the method as explained. You can use the context parameter to define the target context for which the mapping is performed. The `SagCredentials` are sent by the application which calls the login module and therefore, must not be modified. You set the values of the super class variables using the `mapName` method and `mapPassword` method, if applicable.

```
protected mapName(String context, SagCredentials credentials, Map options)
throws SagGeneralSecurityException
```

### Parameters

The following table outlines the parameters of `SimpleNameMappingLoginModule`.

| Parameter | Description |
|---|---|
| `user_mapping_url` | Mandatory only if you use properties file mapping.<br><br>It defines the URL of the Java properties file that contains the mapping information. |
| `user_mapping_regex` | Required only if you use regular expression mapping.<br><br>It defines what regular expression to use to collect the user name from the input name. |
| `user_mapping_matchgroup` | Optional.<br><br>Defines the regular expression group which is used for the results of the regular expression. The default value is `1`. |

### Examples

Example 1:

If you add this LoginModule to the stack:

```
com.softwareag.security.jaas.login.modules.SimpleNameMappingLoginModule required
        user_mapping_url=file://path/to/mapping_user.properties
```

The *mapping_user.properties* file contains the following entries:

```
testclient=Test Client
testclient.password=secret1
```

If you login with username "testclient" the login modules after SimpleNameMappingLoginModule will receive username "Test Client" and password "secret1" as credentials.

Example 2:

If you add this LoginModule to the stack:

```
com.softwareag.security.jaas.login.modules.SimpleNameMappingLoginModule required
        user_mapping_regex="CN=(\\w*),(.*)"
```

If you login with username “CN=Client1, OU=R&D, O=RSUBJET, C=DE” the login modules after SimpleNameMappingLoginModule will receive username "Client1" as credentials.

Example 3:

If you add this LoginModule to the stack:

```
com.softwareag.security.jaas.login.modules.SimpleNameMappingLoginModule required
        user_mapping_regex="CN=(\\w*),(.*)"
        user_mapping_matchgroup="3"
```

If you login with username “CN=Client1, OU=R&D, O=RSUBJET, C=DE” the login modules after SimpleNameMappingLoginModules will receive username null as credentials.

## X509CertificateLoginModule

### Description

You use `X509CertificateLoginModule` to verify one or more than one X.509 certificate. The login module builds all chains of trust and at least one chain must end at the Trust Anchor. All certificates in the chain are verified according to the Public Key Infrastructure extensions (PKIX). The module checks the statuses of the certificates against Certificate Revocation Lists (CRLs). It can import missing certificates from PKCS#7 files. To get the CRL, the validation of the login module supports CRL distribution point (CRL DP). To enable CRL DP, you can set the value of the Java system property `com.sun.security.enableCRLDP` to true. The login module also provides direct trust. This means that the module checks whether the end entity certificate is part of the truststore. If it is, direct trust is created and further CRL checks are disabled.

### Parameters

The following table outlines the parameters of the `X509CertificateLoginModule`. The parameters allow you to extend the login module functionality and plug in other certificate validation methods in it.

| Parameter | Description |
|---|---|
| `truststore_url` | Defines the URL of the keystore which contains the Trust Anchors. This is the RootCA or certificate authority (CA) certificates that are trusted. |
| `truststore_password` | Defines the password of the trust keystore. |
| `truststore_type` | Optional.<br><br>Defines the type of the trust keystore.<br><br>Valid values are:<br><br>■ PKCS7<br>■ PKCS12 |

| Parameter | Description |
|---|---|
| | ■ JKS - Default value. |
| `check_crl_status` | Boolean.<br><br>The following list outlines the possible options:<br><br>■ **true**<br>The status of the end entity certificate is checked against a URL. In this case, the `crl_url` parameter must be set.<br>■ **false**<br>The login module is set to use direct trust. |
| `crl_url` | Mandatory if the `check_crl_status` is set to `true`.<br><br>Defines the URLs of the CRL for the end entity certificate. The URLs are separated by a space. |
| `overwrite_username` | Optional. Boolean.<br><br>Valid values are:<br><br>■ true - Default value. The user name is overwritten with the certificate subject distinguished name (DN).<br>■ false - The module accomplishes only validation of the certificates. |
| `additional_certificates_container_url` | Optional.<br><br>Defines the URL of the container of additional certificates. |
| `additional_certificates_container_type` | Optional.<br><br>Defines the type of the container of additional certificates.<br><br>Valid values are:<br><br>■ PKCS7<br>■ PKCS12<br>■ JKS |
| `additional_certificates_container_password` | Mandatory only if the `additional_certificates_container_type` parameter is set to JKS or PKCS12. |

| Parameter | Description |
|---|---|
| | Defines the password of the certificate container. |

### Example

The following sample excerpt outlines X509CertificateLoginModule and the corresponding configuration that is included in a login context of a JAAS configuration file. The example also shows how the login context reads crl_url, truststore_url, and truststore_password from the Java system parameters. Note that every Java system parameter that is included in the JAAS configuration file must have a value that differs from NULL or the empty string. Failure to do so may cause an exception on the system.

```
/** Login Configuration for the X509CertificateLoginModule **/
X509Login {
    com.softwareag.security.jaas.login.modules.X509CertificateLoginModule required
        check_crl_status=true
        crl_url="${com.softwareag.security.crl.url}"
        truststore_url="${com.softwareag.security.truststore.url}"
        truststore_password="${com.softwareag.security.truststore.password}"
        truststore_type=jks
        overwrite_username=false
        ↩
additional_certificates_container_url="${com.softwareag.security.certificate.container.url}"
        additional_certificates_container_type="jks"
        ↩
additional_certificates_container_password="="${com.softwareag.security.certificate.container.password}";
};
```

## SAMLArtifactLoginModule

### Description

You use SAMLArtifactLoginModule to verify credentials received as SAML artifacts. The module uses the opensaml library and supports SAML version 1.1. It sends a request and validates the SAML artifact against a SAML endpoint, which is the authority issuer of the artifact. The authentication is successful only if the endpoint validates the SAML artifact successfully. The result of the validation is a SAML response that contains information about the owner of the artifact. A part of this response is the user name. If configured in the JAAS configuration file, the login module can overwrite the user name in the SagUserPrincipal with the one that is received in the SAML response.

### Parameters

The following table outlines the parameters of `SAMLArtifactLoginModule`.

| Parameter | Description |
|---|---|
| `saml_identity_provider_url` | Defines the URL of the SAML authority that validates the artifact. |
| `overwrite_username` | Optional. Boolean.<br><br>Valid values are:<br><br>■ true - Default value.<br>■ false - The user name is overwritten with the one that is received in the SAML artifact validation process. |

### Example

The following sample excerpt outlines `SAMLArtifactLoginModule` and the corresponding configuration that is included in a login context of a JAAS configuration file. In this example, the login context reads the `saml_identity_provider_url` parameter from the Java system parameters. Note that every Java system parameter that is included in the JAAS configuration file must have a value that differs from `NULL` or empty string. Failure to do so may cause an exception on the system.

```
/** Login Configuration for the SAMLArtifactLoginModule **/
SAMLArtifactLogin {
    com.softwareag.security.jaas.login.modules.SAMLArtifactLoginModule required
        saml_identity_provider_url="${com.sample.security.saml.samlendpoint}"
        overwrite_username=true;
};
```

## SSXLoginModule

With this login module, you can authenticate users and retrieve groups through the native SSX library.

`SSXLoginModule` is distributed as a template property file within the *sin-ssx.jar* file. It performs authentication using one of the following combination of credentials:

- User name and password
- User name, password and an IAF token (or IAF artifact)
- IAF token (or IAF artifact) only

See *`SSXLoginModule` Configuration Template* for the content of the template file.

**Note:** The template file holds the properties that will be changed rarely, while the configuration file holds the properties that will be changed frequently.

Authentication is done against LDAP, Active Directory, IAF server, or operating system.

In the case of authentication against an IAF server, you may receive an IAF token or artifact during delegated authentication that is added as a *SecurityToken* to the *SharedMap*. This IAF token authenticates you. Authentication is done in the following way:

- The module verifies the token against the IAF server.
- After verification, the module confirms the token's validity and your user identity.

### To use `SSXLoginModule`

1 Set *sin-common.jar* and *sin-ssx.jar* in the classpath.

2 Set the <dlls> in `java.library.path`.

### To configure `SSXLoginModule`

1 Configure `SSXLoginModule` to use RMI

2 Configure and start the RMI server.

There is a set of default parameters for the configuration of `SSXLoginModule`. These parameters are used by default and you can overwrite all of them within the JAAS configuration file.

The information about the `SSXLoginModule` custom parameters is organized under the following headings:

- Parameters for Common Configuration
- Parameters for Internal Repository Configuration
- Parameters for Operating System Configuration
- Parameters for ADSI Configuration
- Parameters for LDAP Configuration

- Parameters for RMI Configuration

## Parameters for Common Configuration

`SSXLoginModule` has several custom parameters for the module options.

Specify your own options, including the mandatory ones that are not included in the template. Your parameters overwrite the ones from the template file.

Security Infrastructure supports a mechanism for overwriting SSX properties which complies with the following pattern:

1. Initially, Security Infrastructure verifies all locally configured settings.
2. If the `OPTIONS_URL` parameter is configured, the functionality attempts to obtain the properties remotely.
   - If the URL address is configured but the respective properties cannot be read and verified, the result is `null` and the login module is disregarded.
   - If the remote properties are verified successfully, the functionality overwrites the respective locally set properties.

     If the remote file provides properties that are not available in the local settings, then the remote properties are taken into account. However, if the remote file does not provide information about any locally set properties, then the functionality preserves the local settings.

The following table outlines the `SSXLoginModule` parameters for common configuration.

| Parameter | Description |
|---|---|
| `options_url` | Optional.<br><br>Defines the URL that specifies the location of the properties file. It may contain any of the listed SSX login module parameters. It allows you to manage all SSX properties centrally.<br><br>No default value.<br><br>**Note:** The parameters in the JAAS configuration file overwrite any parameters in the template properties file. |
| `UseDomainForOptionsURL` | This parameter appends the domain to the value of the `options_url` parameter only if the `options_url` parameter is set, the `UseDomainForOptionsURL` parameter is set to "true", and the user credentials contain a non-empty and non-null domain.<br><br>Valid values are:<br><br>■ true<br>■ false - Default value. |

| Parameter | Description |
|---|---|
| `template_section` | The value is a section from the template authentication properties file.<br><br>**Note:** Only the properties in the specified section are used in the login module.<br><br>Valid values are:<br>■ internal<br>■ os<br>■ adsi<br>■ ldap<br>■ iaf<br><br>No default value. |
| `create_user_principal` | Optional.<br><br>Creates `SagUserPrincipals`.<br><br>Valid values are:<br>■ true - Default value. The `commit ()` method creates the `SagUserPrincipal`. In this case, all existing `SagUserPrincipals` in the `Subject` are removed and replaced by the new one.<br>■ false - The module does not create the `SagUserPrincipal`. In this case, the application is not able to access the user repository. |
| `defaultDomainName` | Optional.<br><br>Specifies the default domain name. If the `defaultDomainName` parameter and the domain name from the `SagCredentials` are not null and not equal, the login module returns "false" in the authentication phase.<br><br>No default value. |
| `CreateGroups` | Optional.<br><br>Specifies whether the login module creates user groups.<br><br>Valid values are:<br>■ true - Default value.<br>■ false - `GroupPrincipals` is not created. |
| `CreateGroupProperties` | Optional.<br><br>Specifies whether the login module creates user group properties.<br><br>Valid values are: |

| Parameter | Description |
|---|---|
| | ■ true - Default value. The login module creates group properties.<br>■ false - The login module does not create group properties. |
| `CreateUserProperties` | Optional.<br><br>Specifies whether the login module creates user properties.<br><br>Valid values are:<br><br>■ true - Default value. The login module creates user properties.<br>■ false - The login module does not create user properties. |
| `propertyMapping.number` | Optional.<br><br>Specifies the number of mapped properties. Use this property if you want to have property mapping.<br><br>A valid value is any positive integer value.<br><br>No default value. |
| `propertyMapping.X.key` / `propertyMapping.X.value` | Required only if `propertyMapping.number` is given.<br><br>Specifies key/value pairs for the property mapping.<br><br>Valid values: X (key and value) runs through all values from "0" to "(`propertyMapping.number` – 1)". For example, "propertyMapping.0.key=<string>".<br><br>No default value. |
| `nativeLogFile` | Optional.<br><br>Specifies the output file for logging.<br><br>No default value. |
| `nativeLogLevel` | Optional.<br><br>Specifies the value of the logging level.<br><br>Valid values are the Integer values from 1 to 6.<br><br>No default value. |
| `cacheTime` | Optional.<br><br>Specifies for how long the authenticated user stays in cache. The value is in seconds.<br><br>A valid value is any Integer value.<br><br>The default value is 180. |

| Parameter | Description |
|---|---|
| `denyTime` | Optional.<br><br>Specifies for how long the authenticated requests of a particular user (userID) are ignored. The value is in seconds.<br><br>A valid value is any Integer value.<br><br>The default value is 100. |
| `denyCount` | Optional.<br><br>Specifies the number of invalid login attempts.<br><br>A valid value is any Integer value.<br><br>The default value is 3. |
| `cacheSize` | Optional.<br><br>Specifies the maximum number of successfully authenticated users that are stored in the cache. When the cache overflows, the oldest entry is removed.<br><br>A valid value is any Integer value.<br><br>The default value is 300. |

### Parameters for Internal Repository Configuration

This section describes the INTERNAL repository type which is based on a text file. The current default repository type, OS, requires specific root privileges on UNIX. To avoid the necessity of specific privileges, it is recommended that you use the internal repository as the default user repository for new installations that use SSX on UNIX.

The internal repository text file is an alternative to the OS and LDAP repositories. It is recommended to use an internal repository only during the initial setup of all required components or until you configure a real repository.

The following table outlines the internal repository mode parameters of the `SSXLoginModule`.

| Parameter | Description |
|---|---|
| `authType` | Specifies the user repository type. The required value is INTERNAL.<br><br>No default value. |
| `internalRepository` | Specifies the path of the internal text repository file.<br><br>For more information, see *Creating Internal User Repository Files*. |
| `defaultDomain` | Optional. |

| Parameter | Description |
|---|---|
| | Specifies a default domain name. When calling the `getAllUsers()` method the `defaultDomain` parameter is added in front of any user ID returned. |

**Example**

The following sample excerpt outlines the INTERNAL mode of the `SSXLoginModule` and the corresponding configuration which is included in a login context of a JAAS configuration file.

```
/** Example Login Configurations for the SSXLoginModule **/
/* This context is for logging in to the internal repository. */
SSXLoginINTERNAL {
    com.softwareag.security.jaas.login.ssx.SSXLoginModule required
        template_section=INTERNAL
        logCallback=true
        internalRepository="/tmp/myrepo/internalRepo";
};
```

### Parameters for Operating System Configuration

The following table outlines the `SSXLoginModule` parameters for operating system configuration:

| Parameter | Description |
|---|---|
| `authType` | Specifies the user database type. The valid value is `os`.<br><br>No default value. |
| `authDaemonPath` | Specifies the explicit path of the privileged daemon process. Specify this parameter if the *sagssxauthd2* executable file is not in the current directory. Valid value is the valid path to the *sagssxauthd2* module.<br><br>No default value.<br><br>**Note:** UNIX only. |
| `defaultGroup` | Optional.<br><br>Specify a default group name here to be returned with any of the group results that are returned by the repository manager.<br><br>A valid value is any valid group name.<br><br>No default value. |
| `defaultDomain` | Optional.<br><br>If this parameter is specified, its value is used at authentication time when domain name is not specified by the user. If a domain name is specified, the value of this parameter is not used.<br><br>A valid value is any valid domain name. |

| Parameter | Description |
|---|---|
| | No default value. |
| `noImpersonation` | Optional. Boolean.<br><br>Specifies how to access data.<br><br>Valid values are:<br><br>■ true - Access is under the account of the running process.<br>■ false - Default value. The data access is under the impersonated user ID of the logged on user.<br><br>**Note:** Windows only. |
| `unixAddMachineName` | Optional. Boolean.<br><br>Specifies the local machine name (on which the user is authenticated). The machine name is added before users and groups; for example, *machine_name*\*user*.<br><br>Valid values are:<br><br>■ true - If set to "true" (and there is no domain field), you are authenticated against the local machine only.<br>■ false - Default value. You are authenticated on the domain that you logged on. |

### Parameters for ADSI Configuration

The following table outlines the `SSXLoginModule` custom parameters for ADSI configuration.

| Parameter | Description |
|---|---|
| `authType` | Specifies the user database type. The valid value is `adsi`.<br><br>No default value. |
| `defaultGroup` | Optional.<br><br>Specify a default group name here to be returned with any of the group results that are returned by the repository manager.<br><br>A valid value is any valid group of users.<br><br>No default value. |
| `defaultDomain` | Optional.<br><br>If this parameter is specified, its value is used at authentication time when domain name is not specified by the user. If a domain name is specified, the value of this parameter is not used.<br><br>A valid value is any valid domain name. |

| Parameter | Description |
|---|---|
| | No default value. |
| `serverHost` | Optional.<br><br>Specifies the name of the server.<br><br>A valid value is any valid server name and any valid IP address.<br><br>No default value. |
| `adsiPersonBindDn` | Optional.<br><br>Specifies the Personal Bind Distinguished Name (DN) for LDAP required for accessing a user. Use it only when all the users that are accessed are under the same node. Do not use it in cases of normal authentication.<br><br>Valid values: for example, "ou=users, ou=germany, dc=eur, dc=sa, dc=com".<br><br>No default value. |
| `adsiGroupBindDn` | Optional.<br><br>Specifies the Personal Bind Distinguished Name (DN) for LDAP required for accessing a group. Use it only when all the groups that are accessed are under the same node. Do not use it in cases of normal authentication.<br><br>Valid values: for example, "ou=groups, ou=germany, dc=eur, dc=sa, dc=com".<br><br>No default value. |
| `adsiForestDn` | Optional.<br><br>Specifies the name of the forest. You use this value when accessing ActiveDirectory.<br><br>Valid values: for example, "dc=myorg,dc=com".<br><br>No default value. |

### Parameters for LDAP Configuration

The following table outlines the `SSXLoginModule` custom parameters for LDAP configuration.

| Parameter | Description |
|---|---|
| `authType` | Specifies the user database type.<br><br>The valid value is `ldap`.<br><br>No default value. |
| `serverHost` | Specifies the name of the server.<br><br>A valid value is any valid server name and any valid IP address. |

| Parameter | Description |
|---|---|
| | No default value. |
| `serverPort` | Optional.<br><br>Specifies the port of the server.<br><br>A valid value is any valid port number.<br><br>The default value is 389. |
| `serverType` | Optional.<br><br>Specifies the type of the server.<br><br>Valid values are:<br><br>■ OpenLdap - Default value.<br>■ ActiveDirectory<br>■ SunOneDirectory<br>■ Novell<br>■ ApacheDS<br>■ Tivoli<br><br>No default value. |
| `personBindDn` | Specifies the Personal Bind Distinguished Name (DN) for LDAP where the authentication information is stored. This value is prefixed with “userIdField=” when running the LDAP authentication.<br><br>Valid values: for example, "ou=users, ou=germany, dc=eur, dc=sa, dc=com".<br><br>No default value. |
| `groupBindDn` | Specifies the Group Root Distinguished Name (DN) for LDAP where the search for group names starts. This value is prefixed with “groupIdField=” when running the LDAP authentication.<br><br>Valid values: for example, "ou=groups, ou=germany, dc=eur, dc=sa, dc=com".<br><br>No default value. |
| `personObjClass` | Optional.<br><br>Specifies the object classes that are contained in the user entries.<br><br>Valid values: <String_Value1>, <String_Value2>, ..., <String_ValueN>.<br><br>The default value depends on the `serverType` parameter:<br><br>■ top, person (OpenLdap)<br>■ top, person, organizationalPerson, user (ActiveDirectory)<br>■ top, person, organizationalperson, inetorgperson (SunOneDirectory) |

| Parameter | Description |
|---|---|
| | ■ top, person, organizationalPerson, ndsLoginProperties (Novell)<br>■ top, person, organizationalPerson (Apache)<br>■ top, person, organizationalPerson,user (Tivoli) |
| `groupObjClass` | Optional.<br>Specifies the object classes that the group entries contain.<br>Valid values: <String_Value1>, <String_Value2>, ..., <String_ValueN>.<br>The default value is dependent on the `serverType` parameter:<br>■ top, groupOfUniqueNames (OpenLdap)<br>■ top, group (ActiveDirectory)<br>■ top, groupofuniquenames (SunOneDirectory)<br>■ top, groupOfUniqueNames (Novell)<br>■ top, groupOfUniqueNames (Apache)<br>■ top, group (Tivoli) |
| `personGrpAttr` | Optional.<br>Specifies the property name of a user entry. It points from a user entry to the group that the user is a member of.<br>Valid values: <String_Value>.<br>The default value depends on the `serverType` parameter:<br>■ ou (OpenLdap)<br>■ memberOf (ActiveDirectory)<br>■ NULL (SunOneDirectory)<br>■ NULL (Novell)<br>■ NULL (Apache)<br>■ memberOf (Tivoli) |
| `groupPrsAttr` | Optional.<br>Specifies the property name of a user entry that points from the group to the users.<br>Valid values: <String_Value>.<br>The default value is dependent on the `serverType` parameter:<br>■ uniqueMember (OpenLdap) |

| Parameter | Description |
|---|---|
| | ■ member(ActiveDirectory)<br>■ uniqueMember (SunOneDirectory)<br>■ uniqueMember (Novell)<br>■ uniqueMember (Apache)<br>■ member (Tivoli) |
| `userIdField` | Optional.<br><br>Specifies the property name that denotes a user entry.<br><br>Valid values: <String_Value>.<br><br>The default value is dependent on the `serverType` parameter:<br><br>■ uid (SunOneDirectory)<br>■ cn (others) |
| `groupIdField` | Optional.<br><br>Specifies the property name that denotes a group entry.<br><br>Valid values: <String_Value>.<br><br>The default value is `cn`. |
| `allowDomainAsBaseBindDn` | Optional. Boolean.<br><br>If the domain name is not specified explicitly and the `defaultDomain` parameter is set, this value is interpreted as `BaseBindDN`.<br><br>Valid values are:<br><br>■ true - the `domainname` parameter is interpreted as a `BaseBindDN` (for example, “ou=People,dc=myorg,dc=com”).<br>■ false - Default value. |
| `personPropAttr` | Optional.<br><br>Specifies the property names that can be accessed for a user entry.<br><br>A valid value is a comma-separated list that contains the property names (<String_Value1>, <String_Value2>, ..., <String_ValueN>).<br><br>The list with property names for a user entry is empty in the following cases:<br><br>■ All specified properties do not exist.<br>■ All specified properties are binary properties.<br><br>The default value is dependent on the `serverType` parameter: |

| Parameter | Description |
|---|---|
| | ■ cn, sn, description, telephoneNumber, seeAlso (OpenLdap)<br>■ cn, displayName, description, mail,telephoneNumber, physicalDeliveryOfficeName, givenName, sn, homeDirectory, ou, cn,description (ActiveDirectory)<br>■ uid, cn, sn, title, description, telephoneNumber, seeAlso, postalAddress, postalCode, postOfficeBox (SunOneDirectory)<br>■ cn, fullName, description, eMailAddress, telephoneNumber, departmentNumber, givenName, sn (Novell)<br>■ cn, description, telephoneNumber (Apache)<br>■ top, person, organizationalPerson, user (Tivoli) |
| `groupPropAttr` | Optional.<br><br>Specifies the property names that can be accessed for a group entry.<br><br>The value is a comma-separated list that contains the property names (<String_Value1>, <String_Value2>, ..., <String_ValueN>).<br><br>The list with property names for a group entry is empty in the following cases:<br>■ All specified properties do not exist.<br>■ All specified properties are binary properties.<br><br>The default value depends on the `serverType` parameter:<br>■ uniqueMember (OpenLdap)<br>■ member (ActiveDirectory)<br>■ uniqueMember (SunOneDirectory)<br>■ uniqueMember (Novell)<br>■ uniqueMember (Apache)<br>■ member (Tivoli) |
| `ldapStartTls` | Optional. Boolean.<br><br>Enforces the usage of secure communication (TLS/ SSL).<br><br>Valid values are:<br>■ true<br>■ false - Default value. |
| `resolveGroups` | Optional. |

| Parameter | Description |
|---|---|
| | Specifies the method for finding the groups of a user using the LDAP authentication type.<br><br>Valid values are:<br><br>■ "CP" - This method uses a computed property field that contains all of the groups (virtually) in the user record.<br>■ "RU" - Default value. The recurse up method looks for a particular field ("personGrpAttr") to find the groups of which the current entry is a direct member.<br>■ "RD" - The recurse down method performs an LDAP search to find all groups that have the particular user as a member. There are no more recursions performed at this time. |
| `computedGroupProp` | Optional.<br><br>Denotes the name of the LDAP property. It is activated if `resolveGroups` is set to "CP".<br><br>Valid values: <String_Value>.<br><br>No default value. |
| `ldapSSLConnection` | Optional. Boolean.<br><br>This parameter denotes the secure communication (with set `serverHost` and `serverPort`) through an LDAP server.<br><br>Valid values are:<br><br>■ true<br>■ false - Default value. |
| `followReferrals` | Optional. Boolean.<br><br>Specifies whether the `SSXLoginModule` must follow referrals or not.<br><br>Valid values are:<br><br>■ true - Default value.<br>■ false |
| `refServerBindingType` | Optional.<br><br>Specifies the kind of binding during "referral following".<br><br>Valid values are:<br><br>■ same_creds - Default value. Uses the same credentials for authentication to the next LDAP server. |

| Parameter | Description |
|---|---|
| | ■ no_creds - Uses anonymous binding to the next server. |
| `referralHopsCnt` | Optional.<br><br>Specifies the count of the referral hops. If this parameter is not specified, the count is unlimited.<br><br>A valid value is any positive integer.<br><br>The default value is unlimited. |
| `useLdapTechUser` | Optional. Boolean.<br><br>Allows you to enable the usage of a technical user.<br><br>Valid values are:<br><br>■ true<br>■ false - Default value. |
| `techLdapUserCredFile` | Mandatory only if you enable the usage of a technical user.<br><br>Specifies the path of the technical user credentials file.<br><br>A valid value is any valid directory and file name on the file system.<br><br>No default value.<br><br>For more information, see *Creating Technical User Credential Files*. |
| `techLdapUserKeyFile` | Optional.<br><br>Specifies the path of the alternative key file.<br><br>A valid value is any valid directory and file name on the file system.<br><br>No default value. |

### Parameters for RMI Configuration

The following table outlines the `SSXLoginModule` custom parameters for RMI configuration.

| Parameter | Description |
|---|---|
| `rmiEnabled` | Optional. Boolean.<br><br>Specifies whether the `SSXLoginModule` must access the SSX through RMI.<br><br>Valid values are:<br><br>■ true - In this case, you must set all the parameters described below. |

| Parameter | Description |
|---|---|
| | ■ false - Default value. |
| `rmiServerAddress` | Optional.<br><br>Specifies the host server for RMI. Use it only if `rmiEnabled` is set to “true”, and if the RMI server is not on the same physical machine, that is, “localhost”.<br><br>A valid value is any valid server address.<br><br>The default value is `localhost`. |
| `rmiServerPort` | Optional.<br><br>Specifies the port that the remote server listens. Use it only if `rmiEnabled` is set to “true”.<br><br>A valid value is any valid server port.<br><br>No default value.<br><br>**Note:** `remoteServerPort` is the deprecated version of this parameter. |
| `throw_exc_missing_remote` | Optional. Boolean.<br><br>Valid values are:<br><br>■ true - If RMI is unreachable or missing, the login module throws an exception.<br>■ false - Default value. If RMI is unreachable or missing, the login module cannot authenticate successfully and does not throw an exception. |

## DelegatedAuthenticationLoginModule

You use this login module to issue IAF tokens based on a previously performed authentication outside the SIN infrastructure.

You need this login module in order to use custom authentication in the application. Based on the established trust, you retrieve an IAF token for authentication.

This module establishes trust between the application using this login module and the IAF server (client SSL certificates validation).

The trust relationship is based on digital signatures. For each delegated authentication request, a signed message (SHA1/RSA) is sent from `DelegatedAuthenticationModule` to the IAF server. The IAF server verifies the message and searches for the signer certificate in a white list.

### Parameters for Configuration

The required custom parameters are passed to an SSX function call and to a corresponding Java method.

To use this module via RMI, extend the `Authenticator` classes.

This login module requires the same configuration parameters as `SSXLoginModule` when using IAF authentication.

The following table outlines the parameters of `DelegatedAuthenticationLoginModule`

| Parameter | Description |
|---|---|
| `keystore_url` | Specifies the URL pointing to the Java keystore to be used for signing.<br><br>A valid value is any valid URL.<br><br>No default value. |
| `keystore_password` | Specifies the password for the keystore to be used for signing. |
| `keystore_type` | Optional.<br><br>Specifies the type of the keystore.<br><br>Valid values are:<br>■ PKCS7<br>■ PKCS12<br>■ JKS - Default value. |
| `signkey_alias` | Optional.<br><br>Specifies the key alias to use for signing. |
| `signkey_password` | Mandatory only if it is not the same as `keystore_password`.<br><br>Specifies the password for the signing key, if it is not the same as the keystore password. |
| `canonical_username_prefix` | Optional.<br><br>String that is to be prefixed to the user name. If not present, nothing is prefixed, and just the domain and user name are passed. |
| `create_user_principal` | Optional. Boolean.<br><br>Valid values are:<br>■ true - Default value. The `commit ()` method creates a `SagUserPrincipal` using the `SSXPrincipal`.<br>■ false |

| Parameter | Description |
|---|---|
| `rmiEnabled` | Optional. Boolean.<br><br>Specifies whether `SSXLoginModule` must access SSX through RMI. The login module does not use RMI by default.<br><br>Valid values are:<br><br>■ true<br>■ false |
| `rmiServerAddress` | Mandatory only if the `rmiEnabled` parameter is set to `true` and if the RMI server is not on the same physical machine, that is, “localhost”.<br><br>Specifies the hostname/IP address of the remote server - “user.sam.ple.com” or “10.3.137.20”. |
| `rmiServerPort` | Required only if the `rmiEnabled` parameter is set to `true`.<br><br>Specifies the port that the remote server listens to - “12345”.<br><br>**Note:** `remoteServerPort` is the deprecated version of this parameter. |
| `rmiSSLEnable` | Mandatory only if the `rmiEnabled` parameter is set to `true`.<br><br>Specifies whether the RMI connection uses SSL or not. |
| `rmiKeyStore` | Mandatory only if the `rmiSSLEnable` parameter is set to `true`.<br><br>Specifies the keystore that must be used during the secure RMI communication. |
| `rmiKeyStorePass` | Mandatory only if the `rmiSSLEnable` parameter is set to `true`.<br><br>Specifies the keystore password. |
| `rmiKeyStoreType` | Mandatory only if the `rmiSSLEnable` parameter is set to `true`.<br><br>Specifies the keystore type. |
| `rmiTrustStore` | Mandatory only if the `rmiSSLEnable` parameter is set to “true”.<br><br>Specifies the truststore that must be used during the secure RMI communication. |
| `rmiTrustStorePass` | Mandatory only if the `rmiTrustStore` parameter is specified.<br><br>Specifies the truststore password. |
| `rmiTrustStoreType` | Mandatory only if the `rmiTrustStore` parameter is specified.<br><br>Specifies the truststore type. |
| `throw_exc_missing_remote` | Optional. Boolean.<br><br>Valid values are: |

| Parameter | Description |
|---|---|
| | ■ true - If RMI is unreachable or missing, the login module throws an exception. ■ false - Default value. If RMI is unreachable or missing, the login module cannot authenticate successfully and does not throw an exception. |

## SSXStopLoginModule

### Description

You use `SSXStopLoginModule` as a dummy replacement of the `XmlServerLoginModule`. If a preceding `SSXLoginModule` authenticates the subject, then the `SSXStopLoginModule` terminates the module chain if it is "sufficient".

The following `LoginModule` methods act as follows:

- `login() method` - always returns "false".
- `commit() method` - if a previous `SSXLoginModule` authenticates the subject, returns "true". Otherwise, returns "false".

The following table contains some user cases:

| User case | login() returns... | commit() returns... | Whole authentication |
|---|---|---|---|
| Unauthenticated subject | false | false | LoginModule ignored |
| Authenticated subject | false | true | pass |

### Parameters

None. Any provided parameters are ignored.

### Example

The following sample excerpt outlines `SSXStopLoginModule`.

```
/* Internal repository login module (SSX) */
    com.softwareag.security.jaas.login.ssx.SSXLoginModule requisite
      template_section="INTERNAL"
      logCallback="true"
      internalRepository="C:/SoftwareAG/common/conf/users.txt"
    /*SSXStopLoginModule*/
    com.centrasite.resourceaccess.junit.SSXStopLoginModule sufficient;
```

## SimpleSSXLoginModule

### Description

You use `SimpleSSXLoginModule` to authenticate against the local operating system only. The module invokes a simple command line utility that sends encrypted user name and password, and returns an encrypted answer that indicates whether the authentication is successful. The module does not control any other information, such as users, groups, or user properties. Also, it does not implement `IUserRepositoryManager`. The module encrypts the message using a TripleDES algorithm. The command line utility authenticates the user against the local operating system using an SSX library.

**Note:** If you use the `SimpleSSXLoginModule` on UNIX-based operating systems, you must set the `authDaemonPath` property in the JAAS configuration file.

### Parameters

The following table outlines the parameters of `SimpleSSXLoginModule`.

| Parameter | Description |
|---|---|
| `LoginUtilityPath` | Defines the full location path to the command line utility that performs the login. |
| `authDaemonPath` | This parameter is mandatory only for UNIX operating systems. It defines the full location path of the privileged daemon process. You must specify a value for the property if the executable *sagssxauthd2* module is not available in the current work directory. |

### Example

The following sample excerpt outlines the `SimpleSSXLoginModule` and the corresponding configuration included in a login context of a JAAS configuration file.

```
/** Login Configuration for the SimpleSSXLoginModule **/
SimpleSSXLogin {
    com.softwareag.security.jaas.login.ssx.SimpleSSXLoginModule required
        LoginUtilityPath=<command_line_utility>;
}
```

## RemoteLoginModule

The implementation of LDAP support in Security Infrastructure allows you to access an LDAP server and browse for fully qualified users anonymously or using technical user credentials. To access a protected LDAP server that does not accept anonymous queries, you can use `SSXLoginModule` directly, or configure and use `RemoteLoginModule` to access the remote authentication service (and then `SSXLoginModule`) in the login context of your JAAS configuration. `RemoteLoginModule` allows you to enforce queries that are executed by the technical user and facilitates the discovery of fully qualified users on the LDAP server. This login module is used on the client side and allows you to protect key information about the technical user from the end user. `RemoteLoginModule` is explicitly part of Security Infrastructure and does not interact with the LDAP server directly.

`RemoteLoginModule` extends `SagAbstractLoginModule` and is part of the `sin-common.jar` package. In its initialize phase, the login module follows a standard way to obtain the available `SagCredentials` objects from `CallbackHandler` and all properties from the JAAS configuration. During the login phase, the login module prepares the communication channel with the remote service and uses the host name and port number that are specified in the properties of the login module. Additional properties ensure the communication over a secure SSL or TLS protocol. When the connection is established, the transmitted message contains the name of the system (IP address or a DNS name), all available credentials (`SagCredentials`), and optionally JAAS configuration properties.

After a successful authentication, the login module receives from the remote service a Subject (which has attached `UserRepositoryManager`) as it is created on the remote side. The subject is stored locally in the login module. If the authentication fails on the remote side, `SagGeneralSecurityException` is received in the login module, which on the other hand is converted to a `LoginException` on the client side. In the commit phase, `RemoteLoginModule` parses the local Subject and extracts from it information about all entries, such as principals, public and private credentials, properties and so on. At a later stage, the login module assigns these entries to the Subject on the client side that is connected with the authentication. An important stage is to attach `IUserRepositoryManager` instance which comes from the remote service. As part of Security Infrastructure, you can combine and use this login module with all available login modules to fulfill any custom authentication scenarios.

- Parameters for Configuration
- Remote Service Overview

## Parameters for Configuration

The following table outlines the parameters of `RemoteLoginModule`.

| Parameter | Description |
|---|---|
| `rmiServerAddress` | Defines the host name of the remote server. A valid value is any valid server name or IP address. |
| `rmiServerPort` | Defines the port number on which the remote server is running. |

**Example**

The following sample excerpt outlines `RemoteLoginModule` and the corresponding configuration included in a login context of a JAAS configuration.

```
/** Login Configuration for the RemoteLoginModule **/
RemoteLoginModuleSSL {
    com.softwareag.security.jaas.remote.login.RemoteLoginModule required
        rmiServerAddress="server"
        rmiServerPort="32415";
};
```

## Remote Service Overview

The implementation of the remote service remains in the existing RMI implementation and additionally comprises two classes that are connected to the remote authentication and repository manager `RemoteLoginServer` and `RemoteRepositoryManagerServer` respectively.

The `RemoteLoginServer` class facilitates the second authentication that is based on the credentials that come from the client side. The entire process includes the following stages:

1. The remote service listens on a specified port and supports SSL/TLS server authentication. The port is configured in the *RemoteSSXServer.config* file. When it receives a communication request, if needed, it validates the SSL/TLS certificates.
2. When the certificates are successfully validated, and the communication channel is enabled, the remote service receives a message which contains the name of the client system (an IP address or a remote system name) all available credentials (`SagCredentials`) and optional JAAS configuration entities. The service extracts the name of the client system and checks whether it is received from a secure client.

3. If the client is secure, the service prepares a follow up authentication. The configuration name for technical user is set in the *RemoteSSXServer.config* file and the JAAS configuration file is set as a system property *java.security.auth.login.config* when the RMI server is running.
4. In case of successful authentication, the service returns to the client a generated Subject. The server subject is saved in a cache with the contexts of the currently authenticated clients. In case of failure, `SagGeneralSecurityException` which contains the root cause is returned.

When the RMI server is running, one more object is created for the remote repository manager. This object is an instance of `RemoteRepositoryManagerServer`. The process includes the following stages:

1. The remote repository manager listens on a specified *RemoteSSXServer.configport* and supports SSL/TLS server authentication.
2. The manager extracts the subject from the cached context and then it retrieves repository manager from the principal in the Subject.
3. The server repository manager is used for each remote repository operation and data transfer to the client.

**Parameters**

To configure the remote service to handle technical user requests, you must configure the following properties in the server configuration file (*RemoteSSXServer.config*).

| Parameter | Description |
|---|---|
| `authPortTech` | Optional.<br><br>Defines the remote technical user authentication port. |
| `repoPortTech` | Optional.<br><br>Defines the remote technical repository manager authenticator. |
| `jaasConfigName` | Optional.<br><br>Defines the JAAS configuration name for technical use.<br><br>**Note:** When the RMI server is started and the `jaasConfigName` parameter is specified, the *java.security.auth.login.config* system property has to be specified with the JAAS configuration file. |
| `accessListFile` | Optional.<br><br>Defines the path to the file that contains the IP addresses or DNS names of the trusted clients. |

**Example**

The following sample excerpt outlines the configuration settings of the remote service, which are included in the *RemoteSSXServer.config* file.

```
port = 31415
authPort = 31416
repoPort = 31417
authPortTech = 31418
repoPortTech = 31419
keyStore = ↩
Y:/java/cvs/SIN/SIN_SSX/test/com/softwareag/security/jaas/remote/ssx/localhost.p12
keyStorePass = 123456
keyStoreType = PKCS12
rmiHostName = my_server
jaasConfigName = SSXLoginOpenLDAPTechUserSSL
```

## RoleLoginModule

### Description

`RoleLoginModule` provides authorization information using the roles/permissions storage. The module is implemented according to the JAAS standards. The current user that is already successfully authenticated by other login modules from the chain, is searched in the storage by the fully qualified name. Also, if any of the previous login modules in the chain provides group membership of the user, this login module looks in the storage for the groups and concatenates permissions assigned to the group to the user's permissions. The login module updates already existing `SagUserPrincipal` with the permissions assigned to the current user (direclty assigned or coming from the groups on which is member). Additionally, `SagRolePrincipal` is created for each role on which the user is member and all of those SagPrincipal objects are attached to the Subject.

**Note:** Permissions are added as properties of `SagUserPrincipal` with key name "permissions".

This module recognizes the following configuration options:

| Parameter | Description |
|---|---|
| `provider_class=my.provider.class` | Optional.<br><br>Full class name of the role provider that should be used.<br><br>The default is FileBasedAuthzStoreImpl. |
| `storage_location="C:/tmp/roles.txt"` | Location of the roles storage.<br><br>For FileBasedAuthzStoreImpl that is the full path to the roles file. |

### Example

Sample Configuration is shown below:

```
Default {
   // SSOS login module for SAML signed assertion validation
   com.softwareag.security.idp.saml.lm.SAMLAssertValidatorLoginModule sufficient;

   // Internal repository login module (java based)
   com.softwareag.security.jaas.login.internal.InternalLoginModule required
       template_section=INTERNAL
       logCallback=true
       internalRepository="C:/SoftwareAG/conf/users.txt"
       create_group_principal=true
       groupRepositoryPath="C:/SoftwareAG/conf/groups.txt";

   // Role repository login module
   com.softwareag.security.authz.store.jaas.login.RoleLoginModule optional
       storage_location="C:/SoftwareAG/conf/roles.txt";
  }; ↩
```

## Single Sign-On Service

The Single Sign-On (SSO) service issues and parses a signed SAML assertion that can be used as a single sign-on and delegation token. The default implementation uses the SAML 2 assertion issuance, however SAML 1.1 version is supported as well.

The bundles required for the SSO service are available within all Software AG Common Platform profiles. The SSO service requires a dynamic configuration properties file in order to work correctly. By default, your installation contains a *com.softwareag.sso.pid.properties* file located under <Software AG_install_directory>/profiles/<profile_name>/configuration/com.softwareag.platform.config.propsloader.

### Dynamic Configuration Settings

The following table outlines the parameters of the SSO service dynamic configuration.

| Parameter | Description |
|---|---|
| `com.softwareag.security.idp.keystore.location` | Specifies the location of the keystore to be used.<br><br>Default value is /common/conf/keystore.jks. |
| `com.softwareag.security.idp.keystore.password` | Optional.<br><br>Specifies the password for the keystore to be used. |
| `com.softwareag.security.idp.keystore.type` | Optional. |

| Parameter | Description |
|---|---|
| | Specifies the type of the keystore.<br><br>Valid values are:<br><br>■ PKCS7<br>■ PKCS12<br>■ JKS - Default value. |
| `com.softwareag.security.idp.keystore.keyalias` | Specifies the key alias to use for signing.<br><br>Default value is ssos. |
| `com.softwareag.security.idp.truststore.location` | Optional.<br><br>Specifies the truststore to be used. |
| `com.softwareag.security.idp.truststore.password` | Mandatory only if the `com.softwareag.security.idp.truststore.location` parameter is specified.<br><br>Specifies the truststore password. |
| `com.softwareag.security.idp.truststore.type` | Mandatory only if the `com.softwareag.security.idp.truststore.location` parameter is specified.<br><br>Specifies the truststore type.<br><br>Valid values are:<br><br>■ PKCS7<br>■ PKCS12<br>■ JKS - Default value. |
| `com.softwareag.security.idp.truststore.keyalias` | Specifies the truststore key alias.<br><br>Default value is ssos. |
| `com.softwareag.security.idp.assertion.lifeperiod` | Specifies the time to live for the issued assertion (in milliseconds).<br><br>Default value is 300. |
| `com.softwareag.security.idp.ehcache.location` | Location in which to cache the configuration used for caching incoming SAML assertions. |

# 6 Developing Login Modules

This chapter shows how to develop new login modules based on Security Infrastructure.

The information is useful for creating new login modules by adapting the pre-defined modules configurations.

The chapter details ways of writing basic `LoginModules` and lists possible scenarios for using SIN security components.

The information is organized under the following topics:

## Writing a Module

The information in this section will help you to develop your own login modules.

All `LoginModules` must extend the `SagAbstractLoginModule`. This class is an abstract superclass for all SIN `LoginModules`. It handles the retrieval of credentials for all derived classes and the handling of the inter-LoginModule SSO. Derived classes have to implement `initConfiguration ()` and `authenticate ()`. Check the Javadoc for details.

**Important:** When you extend the `SagAbstractLoginModule`, do not overwrite the `initialized ()` method. If you need to overwrite it, for example when you use a new Callback and CallbackHandler, invoke explicitly the `super.initialize ()` method instead. This prevents the failure of other SIN-based login modules.

**To write a `LoginModule` using `SagAbstractLoginModule`**

1. Define the parameters for the new module.
2. Extend `SagAbstracLoginModule` with main focus on the implementation of `initConfiguration ()` and `authenticate ()`. The first method gets the incoming parameters from the JAAS configuration file in the following way:

   ```
   String optionValue = (String) options.get(OPTION_VALUE);
   ```

   The second method takes care of the actual authentication of the user. It is called by the `login ()` method from the `SagAbstracLoginModule`. You can modify the user credentials according to the inter-LoginModule SSO.

If you want to implement other methods from the `SagAbstracLoginModule` (logout(), commit(), etc.), it is a good idea to invoke the super method from the parent class at the end.

See `Common Security Scenarios` for ways of using SIN with products from the webMethods suite.

## Common Security Scenarios

SIN functionality covers the existing user scenarios for the webMethods Suite for authentication of users, management of roles, and query of user, role, and group information. The login modules are used for different authentication methods. If you configure them according to your environment requirements, you can implement the desired authentication process for your product.

# 7 Using the LDAP Framework

## Overview

LDAP framework is an OSGi service that uses dynamic configuration properties files for configuring an LDAP directory. The aliases from these dynamic configuration files are used in the JAAS configuration file.

The LDAP configuration behavior depends on the `url` property in the JAAS configuration file. The following behavior patterns exist:

- If the `url` property is set in the JAAS configuration file, but no aliases are set, the LDAP login module uses only the server configured via the JAAS configuration file.
- If the `url` property is not set in the JAAS configuration file, and no aliases are set, the LDAP login module uses all servers configured via the LDAP dynamic configuration.
- If the `url` property is not set in the JAAS configuration file, but aliases are set, the LDAP login module uses only the servers configured via the LDAP dynamic configuration with matching aliases.

## Dynamic Configuration Properties

The default dynamic configurations properties file is available in your installation under <Software AG_install directory>\profiles\<Profile_name>\configuration\com.softwareag.platform.config.propsloader. These properties are used with their default values the first time you start your profile. The dynamic configuration properties files must follow specific naming conventions.

The following table outlines the dynamic configuration properties for all LDAP connections.

| **Parameter** | **Description** |
|---|---|
| `watt.server.ldap.DNescapeChars` | String.<br><br>Specifies which characters to escape when building LDAP queries.<br><br>Valid values: all symbols.<br><br>No default value. |
| `watt.server.ldap.retryCount` | Long. Specifies how much retries can be performed on LDAP connections before giving up.<br><br>A valid value is any positive Long number (including 0).<br><br>The default value is 0. |

| Parameter | Description |
|---|---|
| `watt.server.ldap.DNstripQuotes` | Boolean. Specifies whether to remove quotes when building LDAP queries.<br><br>Valid values are:<br><br>■ true - Default value. The login module removes quotes when building LDAP queries.<br>■ false - The login module does not remove quotes when building LDAP queries. |
| `watt.server.ldap.extendedProps` | String. Specifies the additional JNDI properties to be set.<br><br>No default value. |
| `watt.server.ldap.retryWait` | Long. Specifies how many milliseconds to wait between retries.<br><br>A valid value is any positive Long number (including 0).<br><br>The default value is 0. |
| `watt.server.ldap.doNotBind` | Boolean. Specifies whether the login module should perform an actual binding to LDAP servers.<br><br>Valid values are:<br><br>■ true - The login module does not perform an actual binding to LDAP servers.<br>■ false - Default value. The login module performs an actual binding to LDAP servers. |
| `watt.server.ldap.DNescapePairs` | A pair of strings.<br><br>Specifies whether to escape substitutions. Each time the login module meets the first member of the pair, it replaces it with the second member.<br><br>Valid values are pairs. All string of characters are valid values for the members of the pair.<br><br>No default value. |

| Parameter | Description |
|---|---|
| `watt.server.ldap.DNescapeURL` | Boolean. Specifies whether to escape the URL when building LDAP queries.<br><br>Valid values are:<br><br>■ true - The login module escapes the URL when building LDAP queries.<br>■ false - Default value. The login module does not escape the URL when building LDAP queries. |
| `watt.server.ldap.ignore.serverCertificateValidity` | Boolean. Specifies whether the login module should ignore the error if it uses SSL but the server certificate is expired or not yet valid.<br><br>Valid values are:<br><br>■ true - The login module ignores the error.<br>■ false - Default value. The login module does not ignore the error. |
| `watt.server.ldap.extendedMessages` | Boolean. Specifies whether JNDI should use extended messages.<br><br>Valid values are:<br><br>■ true - JNDI uses extended messages.<br>■ false - Default value. JNDI does not use extended messages. |
| `watt.server.jndi.searchresult.maxlimit` | Long. Specifies the maximal number of results the jndi can return when a search is performed.<br><br>A valid value is any positive Long number (including 0).<br><br>The default value is 0 (no limit). |
| `watt.server.ldap.includeOnlyActiveGroups` | Boolean. This option applies only to Integration Server. It is not used in the LDAP Framework. The login module uses this option to remove from the memory those groups that do not belong to both ACL and LDAP.<br><br>Valid values are: |

| Parameter | Description |
|---|---|
| | ■ true - Default value.<br>■ false |

# III Usage of Pluggable Authentication Module (PAM) on UNIX

# 8 Usage of Pluggable Authentication Module (PAM) on UNIX

The Pluggable Authentication Module (PAM) is a standardized architecture to let third parties carry out authentication requests from applications. PAM allows you to perform OS authentication on UNIX.

## PAM Authentication

To perform OS authentication using PAM, the "sagssxauthd2" module tries to load the client-side PAM library, named `libpam.so`, and the `libcrypt.so` security library (`libsec.so/.sl` on HP-UX), using the `ssxsrv` service.

If `libpam.so` is successfully loaded, the "sagssxauthd2" module performs a PAM authentication.

If `libpam.so` could not be loaded or the PAM authentication fails, the module tries to perform a UNIX user authentication using the password database(s) and the `libcrypt.so` security library. If `libcrypt.so` could not be loaded, an error is returned. If `libcrypt.so` is successfully loaded, the "sagssxauthd2" module calls the `getspnam()` function which looks in the local shadow password user database.

- If `getspnam()` finds the correct user entry, the "sagssxauthd2" module returns "true".
- If `getspnam()` does not find the correct user entry, the "sagssxauthd2" module calls the `getpwnam()` function to read the password. The `getpwnam()` function looks in the local password user database.
  - If `getpwnam()` finds the correct user entry, the "sagssxauthd2" module returns "true".
  - If `getpwnam()` fails, the user is rejected due to an invalid password.

## Conditions for Using PAM

Most PAM modules and both `getspnam()` and `getpwnam()`require specific privileges from the calling process. Therefore, "sagssxauthd2" must be owned by the "root" user. Also, the "sagssxauthd2" module must be on a device not mounted with the "nosuid" option and the `setuid` flag must be enabled (the file access rights should look like "-rwsr-sr-x …. root … sagssxauthd2").

If any of the conditions above is not met, an error can occur. In this case, it is important to double-check the status of "sagssxauthd2" and create an SSX trace to be sent to support.

Another source of failure is using an unsupported by SSX hash algorithm for comparing the passwords returned from `getspnam()` and `getpwnam()`. The supported hash algorithms are:

- DES
- MD5
- Long Blowfish

- Short Blowfish
- SHA-256
- SHA-512

**Note:** On HP-UX, the "sagssxauthd2" module also uses the `crypt2_passwd_match()` and `bigcrypt()` functions to perform the comparison.