IBM Security Identity Manager
Version 7.0.1.10

Reference Topics

IBM Security Identity Manager
Version 7.0.1.10

Reference Topics

Table of contents

Table list

Chapter 1. Application extensions

Application extensions can be defined in Java™ class files and run from a workflow.

Application extensions are typically defined in one or more Java class files. They are used when a set of functions needs to be run from a workflow. The functions can be implemented to receive input parameters from a workflow and return parameters back to the workflow.

WorkflowApplication interface

Application extensions that require access to the current workflow context information must implement the `WorkflowApplication` interface.

If the extension does not require any workflow context information, implementing this interface is not required. The following example is a code snippet for implementing the `WorkflowApplication` interface. For a complete example, see the information in the `extensions.zip` file.

From the Appliance Dashboard on the IBM® Security Identity Manager virtual appliance console, click **Configure** > **Advanced Configuration** > **Custom File Management**. From the **All Files** tab, go to `directories/utilities` and download the `extensions.zip` file and extract it.

```
public class CustomEmail implements WorkflowApplication {
public CustomEmail() {
}
```

When you implement the `WorkflowApplication` interface you must define a `setContext` method that accepts a `WorkflowExecutionContext` object. Store this object in a member variable in the implementing class.

```
// The context of the workflow. Passed in from the workflow engine
protected WorkflowExecutionContext ctx;
/**
* Passes the workflow execution context to the application.
*
* @param context WorklowExecutionContext holding information about the
* currently executing activity.
*/
public void setContext(WorkflowExecutionContext ctx) {
this.ctx = ctx;
}
```

Application extension methods

The application can contain whatever processing is required to accomplish the task. An extension can contain any number of methods that can be exposed to the workflow.

The following example is a code snippet of a method that is available in the workflow for the extension node. For a complete example, see the information in the `extensions` directory.

From the Appliance Dashboard on the IBM Security Identity Manager virtual appliance console, click **Configure** > **Advanced Configuration** > **Custom File Management**. From the **All Files** tab, go to `directories/utilities` and download the `extensions.zip` file and extract it.

```
/**
* Method sendMailByProperty.
* This method is called to send an e-mail to an e-mail address specified by
the
* "recipient" property in the specified property file.
* @param person - the requestee's person object
* @param mailTag - the mailtag for this message. Used to look up properties
* @param propertyFileName - the name of the property file that contains
* this message's data
* @param attrList - an optional list of tag/value pairs
* @return ActivityResult - a workflow activity result object
*/
public ActivityResult sendMailByProperty(Person person,
String mailTag,
String propertyFileName,
String attrs) {
String recipient_email = "";
try {
processSendMail(person,mailTag,propertyFileName,recipient_email,
attrs);
return new ActivityResult(ActivityResult.STATUS_COMPLETE,
ActivityResult.SUCCESS,
"Sent Mail",
null);
} catch (CustomEMailDataException e) {
return new ActivityResult(ActivityResult.STATUS_COMPLETE,
ActivityResult.FAILED,
e.getMessage(),
null);
}
}
```

Application Extension methods can receive inputs from the workflow. The inputs defined in the workflow extension window maps to the method arguments (ensure that the types match). The `sendMailByProperty` method returns an `ActivityResult` object. This method allows the application to communicate back to the caller a status and a return value, if necessary. The `ActivityResult` object has member variables for status (int), summary, (String), detail (List), and description (String). Return values are in the detail list. The order of the values in the list must correspond to the order of the output parameters as defined in the extension window. See the IBM Security Identity Manager API documentation for a complete description of the `ActivityResult` class.

Registering extensions

For the workflow to make the extension available with the extension node, it must first be registered in the `workflowextensions.xml` file. From the **Appliance Dashboard** of the IBM Security Identity Manager virtual appliance console, select **Configure** > **Advanced Configuration** > **Library and Workflow Extension**. Use the Library and Workflow Extension page to register the file.

Each method requires an activity entry in the XML file. The activity entry includes these aspects:

Activity ID

An activity ID is required and must be unique in the workflow. This name is in the extension window activity type menu.

Implementation type

The implementation type contains the class name and the method name that is started by this extension.

Parameters sections

The parameters sections list the input and out parameters and their data types. These parameters are in the extension window Input/Out Parameters.

Transition restriction

The transition restriction defines the join type. Split type can also be defined. For more information, see the information in the `extensions.zip` file.

From the Appliance Dashboard on the IBM Security Identity Manager virtual appliance console, click **Configure** > **Advanced Configuration** > **Custom File Management**. From the **All Files** tab, go to `directories/utilities` and download the `extensions.zip` file and extract it.

```
<ACTIVITY ACTIVITYID="SendMailByProperty" LIMIT="600000">
   <IMPLEMENTATION_TYPE>
<APPLICATION
      CLASS_NAME="com.ibm.itim.CustomWorkflowExtensions.CustomEmail"
      METHOD_NAME="sendMailByProperty"/>
   </IMPLEMENTATION_TYPE>
   <PARAMETERS>
<IN_PARAMETERS PARAM_ID="inperson" TYPE="Person"/>
<IN_PARAMETERS PARAM_ID="mailtag" TYPE="String"/>
<IN_PARAMETERS PARAM_ID="propertyfilename" TYPE="String"/>
<IN_PARAMETERS PARAM_ID="attributelist" TYPE="String"/>
   </PARAMETERS>
   <TRANSITION_RESTRICTION JOIN="XOR"/>
</ACTIVITY>
```

The Application Extension class file must be in a JAR file. Do these steps:

1. From the top-level menu of the **Appliance Dashboard**, select **Configure** > **Advanced Configuration** > **Library and Workflow Extension** to display the Library and Workflow Extension page.
2. Click **New** to add an external library.
3. In the Add External Library window, upload the JAR file.
4. Paste the extension information in the **Extension** field.
5. Click **Save Configuration**.
6. From the **Server Control** widget on the **Appliance Dashboard**, restart the server before the extensions are available in the workflow.

Chapter 2. Application programming interfaces

Application programming interfaces (APIs) are part of a plug-in model that you can use to add applications without disrupting existing applications.

Remote application programs run outside of the IBM Security Identity Manager server Java virtual machine (JVM). Classes outside of the application packages are not intended to be started by a remote application. Classes in remote applications are documented under the IBM Security Identity Manager application packages. Server extensions, which run in the IBM Security Identity Manager server JVM, can use any of the classes listed in the published API documentation (Javadoc). They are Java classes that run in the same JVM of the caller. These APIs are used to develop IBM Security Identity Manager customization and extensions that can plug into IBM Security Identity Manager.

Several application APIs can be started by a remote application. A few server extension APIs in the `dataservices` package are included also. The following application APIs are intended to be started by a remote application:

Provisioning Policy API
: Can search, add, modify, and delete provisioning policies in IBM Security Identity Manager from a remote application.

Group API
: Can search, add, modify, and delete anIBM Security Identity Manager group.

ACI API
: Can search, add, modify, and delete an access control information list (access right), but it does not verify authorization.

Reconciliation API
: Can get, add, modify, and delete a reconciliation schedule for a particular service and triggers reconciliation.

The following server extension APIs are included:

- `com.ibm.itim.common.ComplexAttributeValue`
- `com.ibm.itim.dataservices.model.ComplexAttributeHandler`
- `com.ibm.itim.dataservices.model.domain.access.Access`
- `com.ibm.itim.dataservices.model.domain.access.ProvisioningConfiguration`
- `com.ibm.itim.dataservices.model.domain.access.NotificationOption`

Applications API

Use the applications API to create customized or alternative user interfaces. This API provides an interface to the IBM Security Identity Manager provisioning platform.

The applications API provides a set of Java classes that abstract the more frequently used functions of the provisioning platform. Examples are identity management, password management, and account management. The classes that make up this API are the same classes that IBM Security Identity Manager uses for its user interface.

For more information, see the documentation that is provided with the Applications API. Do these steps:

1. Log on to the IBM Security Identity Manager virtual appliance console to open the **Appliance Dashboard**.
2. From the top-level menu of the **Appliance Dashboard**, select **Configure** > **Advanced Configuration** > **Custom File Management** to display the Custom File Management page.
3. Click the **All Files** tab.
4. Go to `directories/utilities`.
5. Select `extensions.zip` and click **Download**.
6. Extract the `extensions.zip` file.
7. Go to `/extensions/`*`version_number`*`/doc/applications`. For example, *version_number* is `7.0`.

For sample codes, see `/extensions/`*`version number`*`/examples/apps`.

Self registration API

Part of the applications API, the self registration API provides an interface to create a person in the provisioning platform without a user context.

The self registration API can be called without a user context. It is set up to start without accessing the system with an IBM Security Identity Manager account login and password. The Self Registration API is part of a customizable process. The process provides an example JavaServer Pages (JSP) page as a product extension based on the default `inetOrgPerson` class. The JSP calls the Self Registration API to create a user.

Access control information list (ACI) API

The ACI API provides an interface for managing the IBM Security Identity Manager access control list, container-by-container.

A remote client can use basic add, list, modify, and delete operations for managing the access control list. However, the ACI API cannot verify authorization to the user.

This API exists in the `com.ibm.itim.apps.acl.AccessControlListManager` class.

IBM Security Identity Manager group API

The IBM Security Identity Manager group API provides system group management capabilities, namely APIs to manage groups on the IBM Security Identity Manager service and groups on managed services. The APIs also provide search capabilities for these groups.

The IBM Security Identity Manager group API provides an interface for managing the groups on either the IBM Security Identity Manager service or on other managed services. You can search, add, modify, or delete these groups. You can also add and remove users in a group on either the IBM Security Identity Manager service or on a managed service.

For groups on the IBM Security Identity Manager service, the API exists in the following classes:

- `com.ibm.itim.apps.system.SystemRoleManager`
- `com.ibm.itim.apps.system.SystemRoleMO`

- com.ibm.itim.apps.system.SystemUserMO

For groups on a managed service, the API exists in the following classes:

- com.ibm.itim.apps.provisioning.GroupManager
- com.ibm.itim.apps.provisioning.GroupMO

Provisioning policy API

The IBM Security Identity Manager provisioning policy API provides an interface to manage provisioning policies that are defined in IBM Security Identity Manager.

This API can search, add, modify, and delete provisioning policy. The API exists in the following classes:

- com.ibm.itim.apps.policy.ProvisioningPolicyManager
- com.ibm.itim.apps.policy.ProvisioningPolicyMO

Recertification policy API

The IBM Security Identity Manager recertification policy API provides an interface to manage recertification policies that are defined in Security Identity Manager.

This API provides capabilities to search, add, modify, delete, and run recertification policies.

The following classes or interfaces are exposed to provide recertification policy management capabilities through APIs.

1. **Core classes:**
   - com.ibm.itim.apps.policy.RecertificationPolicyManager
   - com.ibm.itim.apps.policy.RecertificationPolicyMO
   - com.ibm.itim.dataservices.model.policy.recert.RecertificationPolicy
2. **Dependent classes:**
   - com.ibm.itim.dataservices.model.policy.recert.RecertificationParticipant
   - com.ibm.itim.dataservices.model.policy.RoleTarget
   - com.ibm.itim.dataservices.model.policy.GroupTarget
   - com.ibm.itim.dataservices.model.policy.ServiceTarget
   - com.ibm.itim.scheduling.RecurringTimeSchedule
3. **Abstract classes extended by recertification policy directly or indirectly:**
   - com.ibm.itim.dataservices.model.policy.DirectoryPolicy
   - com.ibm.itim.dataservices.model.policy.ScopedPolicy
   - com.ibm.itim.dataservices.model.policy.ServicePolicy
4. **Interface implemented by recertification policy or dependent classes directly or indirectly:**
   - com.ibm.itim.dataservices.model.policy.Policy
   - com.ibm.itim.dataservices.model.policy.IPolicyTarget
   - com.ibm.itim.scheduling.Schedulable

Reconciliation API

The reconciliation API can create and query reconciliations and reconciliation parameters.

The Reconciliation API provides an interface to manage reconciliation schedules of services. You can:

- Get and set reconciliation schedules to a service.

- Modify the reconciliation schedules collection, which includes additions and deletions.
- Set the new collection.
- Trigger a specific reconciliation schedule to run.

The API exists in the following classes:

- `com.ibm.itim.apps.recon.ReconManager`
- `com.ibm.itim.apps.recon.ReconUnitData`

Authentication API

Use the authentication API for working with different trusted identity stores such as identity information. This information can be stored on a Windows domain server or an LDAP directory. It includes the use of different types of keys, typically passwords, to unlock the application for a user.

The authentication API contains the authentication client API, which makes authentication requests, and the authentication provider API, which implements authentication requests.

Data services API

The data services API provides an interface to the IBM Security Identity Manager data model.

This API abstracts the more commonly used data model entities such as identities, accounts, access, and services in the provisioning process. It includes a generic interface to handle complex attributes. Data synchronization depends on Data Services APIs. Furthermore, the data services API provides the data model that the Applications API uses.

Although the ability to change the data model is provided in this API, this ability is not its focus. The Data Services API is low level. It abstracts the physical layout of the data store (directory structure). It does not provide the business logic that the provisioning applications with the platform provide.

IBM Directory Integration API

With this API, IBM Security Directory Integrator can import identity information into IBM Security Identity Manager. It manages accounts in the IBM Security Identity Manager data store on external resources that use IBM Security Directory Integrator.

The following features are included in this API:

Note: Directory Service Markup Language version 2 (`DSMLv2`) was deprecated.

- A Directory Service Markup Language version 2 (`DSMLv2`) `ServiceProvider`. You can use it to import data. IBM Security Identity Manager acts as a `DSMLv2` client. IBM Security Directory Integrator acts as a `DSMLv2` server.
- A `DSMLv2` event handler. You can use it to import data into IBM Security Identity Manager. IBM Security Identity Manager acts as a `DSMLv2` server. IBM Security Directory Integrator acts as a `DSMLv2` client.

- Ready-to-use schema support for communicating with IBM Security Directory Integrator. You can use IBM Security Directory Integrator as an endpoint and define it as a service instance in the IBM Security Identity Manager user interface for identity feed.

JavaScript API

The JavaScript API extends the scripting components that are specific to the scripting language that is configured with the product.

IBM Security Identity Manager provides a method to register new JavaScript extensions with the server. You can use the JavaScript API to add additional objects and functions to the interpreter's glossary. A client can create and register additional objects and functions with the interpreter to run at run time.

The JavaScript API provides information about access participants, such as participant type, workflow participants, group access management, and access notification context.

Mail API

Use the mail API to customize mail content, format, and notification recipients.

Clients who use this API can make notification requests and extend construction of notification messages. The Mail API contains the Mail Client API, which makes notification requests, and the Mail Provider API, which implements notification requests.

The mail API also contains a function that is called Post Office that prevents workflow participants from receiving multiple email notifications that have similar content. Similar emails are stored, combined into a single email notification, and forwarded to a user.

Password rules API

The password rules API provides an interface to customize the standard password rule set and random password generation process.

You can use the password rules framework to customize the mechanism of generating passwords by the IBM Security Identity Manager server. Use one of the following ways to add custom logic to the password framework:

- A custom rule
- A custom generator
- Custom rules and a custom generator

Policy analysis API

The policy analysis API provides an interface to information about policies that are defined in the IBM Security Identity Manager Server. It is an interface to the access granted to a specific individual.

The API contains a set of Java classes that retrieve and abstract the provisioning policy information that controls access to managed resources. The Provisioning Policy API reports the provisioning policy enforcement in the system, but it does

not support client modification of the policy. A client can use the policy information for auditing or deciding about potential policy enforcement changes.

Service provider API

The service provider API provides custom connectors. The connectors can be used from the IBM Security Identity Manager provisioning platform or any other Java-based provisioning platform that supports the same interface.

Service provider APIs define the interface that the IBM Security Identity Manager adapter needs to implement and communicate to remote adapter agents. The adapter agent implementation does not rely on IBM Security Identity Manager APIs except for the set of asynchronous notification APIs provided under Service Provider APIs.

The following operations are included in the interface between the provisioning platform and the connector:

- Add
- Change password
- Delete
- Modify
- Restore
- Search
- Suspend
- Test

The provisioning platform performs all of the operations needed to determine the actions and their parameters that are to be run against resources. The connector runs those operations on the resource within requirements that are related to the resource.

Single sign-on API

The single sign-on API provides a single sign-on interface to accessible resources.

Some IBM Security Identity Manager installations might require integration with third party, single sign-on providers. Typically, such single sign-on providers protect a set of web-based resources with an authentication data store that is managed separately from IBM Security Identity Manager. The first time a client attempts to access any protected resource, the single sign-on provider provides authentication. If access is granted, the provider passes a token that indicates the identity of the authenticated user to all resources that are accessed later.

Web services API

This API consists of multiple web services, which are grouped by function. The services are listed alphabetically except the `WSSessionService`. This service is listed first since it is the first service that is called by any application. The session object that is returned by its login method is used as a parameter in all subsequent services.

WSSessionService

The WSSessionService web service provides authentication, session creation, and password challenge authentication. A client calls WSSessionService before you start any other web services. WSSessionService returns a session (handle) object that must be passed to the other web service calls to maintain a threaded conversation. The service provides the following operations:

- Login.
- Logout.

You can also use the WSUnauthService web service for other operations.

WSAccessService

The WSAccessService web service provides the following operations:

- Create a user access.
- Retrieve existing user access of a person.
- Remove user access.
- Search access entitlements available to a person.

The service provides following operations:

- Create and modify accesses.
- Do access searches.

WSAccountService

The WSAccountService web service provides the following operations to do account-related tasks:

- Create, modify, and other simple account operations.
- Retrieve default account attributes for a new account as specified by the provisioning policy.
- Retrieve the account profile name for a service.

WSExtensionService

The WSExtensionService web service provides a framework to extend the existing web services that are used by users. The service provides the users to create an operation to show a new Security Identity Manager API. The detailed steps to create an extension service are specified in the ITIMWS.odt file, which is in the extensions.zip file.

From the Appliance Dashboard on the IBM Security Identity Manager virtual appliance console, click **Configure** > **Advanced Configuration** > **Custom File Management**. From the **All Files** tab, go to directories/utilities and download the extensions.zip file and extract it. View the ITIMWS.odt file in \extensions\7.0\doc\ws.

WSGroupService

The WSGroupService web service provides group management functions. The service provides the following operations:

- Create and remove groups.
- Search groups.

- Manage group membership.

WSOrganizationalContainerService

The `WSOrganizationalContainerService` web service provides Security Identity Manager organization tree traversal and retrieval methods.

WSPasswordService

The `WSPasswordService` web service provides password management functions. The service provides the following operations:

- Validates the password as per the password policy rules.
- Enables change or generate password.

WSPersonService

The `WSPersonService` web service provides person-object related methods. The service provides the following operations:

- Create, modify, suspend, restore, delete, and other simple person operations.
- Retrieve the services to which a person is entitled in Security Identity Manager or accounts.
- Do person searches.
- Retrieve the person object of the Principal.

WSProvisioningPolicyService

The `WSProvisioningPolicyService` web service deals with the provisioning policy. The service provides the following operations:

- Search provisioning policies.
- Create, modify, and delete provisioning policies.

WSRequestService

The `WSRequestService` web service provides the Security Identity Manager request related functions. The service provides the following operations:

- Search for completed requests.
- Retrieve pending requests.
- Retrieve the request object that is based on the process ID or request ID.

WSRoleService

The `WSRoleService` web service provides role-based capabilities in the Security Identity Manager. The service provides the following operations:

- Create and modify roles.
- Do role searches.
- Manage role hierarchy.

WSSearchDataService

The `WSSearchDataService` web service provides functions to search various Security Identity Manager directory objects. The search method does not enforce the

Security Identity Manager ACIs, but a valid Security Identity Manager session is required to call these methods. The service provides the following operations:

- Search for persons from `root` container.
- Search for persons that are having an Security Identity Manager account.
- Search for the possible delegates within Security Identity Manager for the logged-in user.
- Retrieve the searchable attributes of an entity in Security Identity Manager.
- Retrieve common searchable attributes for the Security Identity Manager entity.

WSServiceService

The `WSServiceService` web service provides Security Identity Manager-based managed services (end-point configuration) functions. The service provides the following operations:

- Retrieve support data. For example, group data for UNIX, Linux, or Microsoft Windows services.
- Determine whether a password is required when provisioning on a service.
- Retrieve services that are configured on Security Identity Manager.

WSSharedAccessService

The `WSSharedAccessService` web service provides many functions for the shared access module that is introduced in Security Identity Manager . The web service clients must call the login method before it calls any other web services. The service provides the following operations:

- Retrieve authorized shared accesses.
- Retrieve the credentials.
- Check in or checkout credentials.

Note: You must install and enable the shared access module in order to use the WSSharedAccessService API.
For more information, see Shared access web services API.

WSSystemUserService

The `WSSystemUserService` web service provides the functions that are related to system users. The service provides the following operations:

- Manage delegates, that is, add, modify, or delete delegates.
- Retrieve all the system roles.
- Configure challenge response.
- Search for system users who have an Security Identity Manager account.

WSToDoService

The `WSToDoService` web service provides the functions to manage the different activities available in Security Identity Manager. The service provides the following operations:

- Approve or reject activities.
- Retrieve or Submit Request for information activity details.
- Retrieve the pending activities of the logged-in user.

WSUnauthService

The `WSUnauthService` web service provides an interface for all the web service APIs that do not require the Security Identity Manager authentication. The service provides the following operations:

- Version information.
- Reset password by using the challenge responses.
- Password policies.

Updates to the web services application interface programming (API)

There are updates available for IBM Security Identity Manager web services APIs in any fix pack that is later than IBM Security Identity Manager 7.0.0.2.

Web services APIs for the person search, which belongs to the `WSPersonService` and `WSSearchDataService` services are updated to enable the search person functionality for any person category, such as `Person`, `BPPerson`, or `Custom person`.

The new web service `WSRoleService` deletes an organizational role.

For detailed information about web services APIs, see the `ITIMWS.odt` file. To access this file, do these steps:

1. From the **Appliance Dashboard** on the IBM Security Identity Manager virtual appliance console, click **Configure** > **Advanced Configuration** > **Custom File Management**.
2. From the **All Files** tab, go to `directories/utilities`
3. Select the `extensions.zip` file
4. Click **Download** and extract the file.
5. In the extracted directory, go to `extensions/7.0/doc/ws`.
6. View the `ITIMWS.odt` file.

For information about how to migrate existing APIs to IBM Security Identity Manager web services, see the `OPALWebServicesMigrationGuidelines.doc`. To access this file, go to `extensions/7.0/doc/ws/migration`.

Workflow API

Use the workflow API for custom code that can be called from a workflow process as a custom Java application or a JavaScript function. This custom code can then do special business logic, query external data stores, or provide integration with other workflow engines.

The Workflow API consists of a set of Java classes. The classes abstract the more commonly used concepts of the workflow environment, such as processes, activities, and relevant data.

The Workflow API supports new access request types. The access owner is a participant type.

The Workflow API provides methods for updating the recertification state and provides audit information for recertification. Audit records contain information about the recertification configuration and the *who*, *what*, and *when* of recertification tasks. These audits provide more useful reports about recertification compliance of

users, accounts, and accesses. Consumers of the recertification policies can also have their recertification process audited in a reportable way.

Chapter 3. Dictionary for a password policy

You can create a dictionary for a password policy rule that rejects certain terms as passwords.

To use a dictionary for a password policy rule, you must first create and load a `dictionary.ldif` file to the IBM Security Identity Manager Server. To create a dictionary for a password policy rule:

1. Using an ASCII or other plain text editor, create a dictionary that contains the list of terms in an LDAP Data Interchange Format (LDIF) file.

   For example, create a file similar to this `dictionary.ldif` file, which specifies the domain as `dc=com`:

   ```
   dn: erword=test,erdictionaryname=password, ou=itim, dc=com
   erWord: test
   objectclass: top
   objectclass: erDictionaryItem

   dn: erword=secret,erdictionaryname=password, ou=itim, dc=com
   erWord: secret
   objectclass: top
   objectclass: erDictionaryItem

   dn: erword=password,erdictionaryname=password, ou=itim, dc=com
   erWord: password
   objectclass: top
   objectclass: erDictionaryItem
   ```

2. Use an LDAP browser to import the `dictionary.ldif` file on to the IBM Security Identity Manager Server.

The dictionary file can now be used in the password strength rule.

Chapter 4. Dataservices attributes for recertification

IBM Security Identity Manager provides optional attributes in the erAccountItem object class to represent different values for recertification.

Overview

The dataservices attributes for recertification are relevant only if recertification is enabled for specific accounts or accesses.

The following optional attributes are provided:

- erLastCertifiedDate
- erRecertificationLastAction
- erAccessLastCertifiedDate
- erAccessRecertificationLastAction

erLastCertifiedDate

The erLastCertifiedDate attribute is updated by the account recertification process only, but not for accesses. An optional attribute for the timestamp of the last time the account was marked as recertified. This attribute is updated on approved recertifications regardless of recertification policy schedule type, whether rolling or calendar style.

This attribute is updated for both approvals during normal recertification cycle and through the recertificationOverride option outside of the normal recertification policy run. The absence of a value means that recertification was never approved for this account. The Account data services object from the com.ibm.itim.dataservices.model.domain package defines the setLastCertifiedDate() and getLastCertifiedDate() methods for accessing this attribute. When an account is *certified*, this attribute must be updated along with reRecertificationLastAction.

erRecertificationLastAction

The erRecertificationLastAction attribute is updated by the account recertification process only, but not for accesses. This attribute requires a getter and setter method defined on the Account data services object class com.ibm.itim.dataservices.model.domain package:

```
public void setRecertificationLastAction(String recertificationAction)
public String getRecertficiationLastAction()
```

This optional attribute describes the action taken the last time recertification was run. The following values are valid:

- com.ibm.itim.dataservices.model.domain.Account.CERTIFIED = 'CERTIFIED'
- com.ibm.itim.dataservices.model.domain.Account.CERTIFIED_ADMIN = 'CERTIFIED_ADMIN'
- com.ibm.itim.dataservices.model.domain.Account.REJECTED_MARK = 'REJECTED_MARK'
- com.ibm.itim.dataservices.model.domain.Account.REJECTED_SUSPEND = 'REJECTED_SUSPEND'

erAccessLastCertifiedDate

The erAccessLastCertifiedDate attribute is specific to accesses that are defined on an account. This multivalued attribute holds the access group definition distinguished name and timestamp that shows when that access was last certified as a delimited string.

Example

```
eraccesslastcertifieddate:  erntlocalname=users,
erglobalid=7281584268561021074,ou=services,
erglobalid=00000000000000000000,ou=hawk,o=ibm,
c=us;;200711202115Z
```

This example shows the last recertification date for the access that is associated with the access defined for the group specified by the distinguished name. Only one value for this attribute per access is defined for the account.

erAccessRecertificationLastAction

The erAccessRecertificationLastAction attribute is specific to recertification state of accesses that are defined on an account. This multivalued attribute holds the access group definition distinguished name and recertification last action taken as a delimited string. It serves the same purpose for accesses as erRecertificationLastAction does for accounts.

Example

```
eraccessrecertificationlastaction:  erntlocalname=users,
erglobalid=7281584268561021074,
ou=services,erglobalid=00000000000000000000,
ou=hawk,o=ibm,c=us;;CERTIFIED
```

This example shows the last recertification action for the access that is associated with the group definition distinguished name. The values for the action are the same as described for the erRecertificationLastAction attribute. Only one value for this attribute per access is defined for the account.

Chapter 5. Date range customization

IBM Security Identity Manager provides additional date range customization, which is not available through the standard Form Designer applet.

With these options, you can control the years available to users when they customize a date. The following options must be configured manually on the following form template that is stored in the directory server:

`erformname=inetOrgPerson,ou=formTemplates,ou=itim,ou=tivsys,dc=com`

You can specify options that define the range of years to be displayed. You also can specify the standard range of years, a special extended max year such as 9999, or special minimum value such as 1900. You have options to display all years between the standard range and extended dates.

The options are:

minYear
: Minimum year to display.

spanMinYearRange
: When set to a value of false, displays all years between `minYear` and `minRangeYear`.

minRangeYear
: Starting year for the standard range of years. The default is 1990.

maxRangeYear
: Ending year for the standard range of years. The default is 2010.

spanYearRange
: When the value is false, displays all years between `maxRangeYear` and `maxYear`.

maxYear
: Maximum year to display.

Chapter 6. Workflow extensions

Workflow extensions provide a means to alter or extend workflow functions.

Policy enforcement extension

The policy enforcement extension assesses the accounts that are associated with a Person or BPPerson and enforces the policies in place for that person.

Overview

A policy enforcement extension is code that can be called directly from a workflow. Workflows that change a person object typically use this extension.

The extension is implemented in `com.ibm.itim.workflowextensions.PersonExtensions`.

The following extensions are provided:

- `enforcePolicyForPerson(Person, skipNonEntitledAccountsEvaluation)`
- `enforcePolicyForPerson(BPPerson, skipNonEntitledAccountsEvaluation)`

The extensions work identically on the specified `Person` or `BPPersion`.

`skipNonEntitledAccountsEvaluation` is a string, either `true` or `false`.

- If `false`, then all accounts applicable to the person are evaluated. All accounts that the person owns are considered when the extension enforces provisioning polices.
- If `true`, then policy enforcement proceeds as follows:
  1. Identify all services applicable for the person store them in a collection.
  2. Check for removed roles in the change list of the specified person.
  3. Merge the list of services that are identified in step 1 and step 2.

     This process specifies that only accounts calculated from the person's role change are considered for policy enforcement. No other accounts are considered.

     Therefore, some accounts are not considered: accounts where the person's role is removed, and accounts that are already provisioned for those roles.

For examples of how the extensions are used, see the Add, Modify, and Transfer operations in Operations management.

Recertification extensions

The recertification extensions track the recertification state in a workflow.

Overview

A recertification extension is code that can be called directly from a workflow. An extension defined for accounts also handles the recertification state for accesses, and uses `dataservices` to update attributes stored on the account object in data

services. These extension methods are integrated into the AccountExtensions class from the com.ibm.itim.workflowextensions package.

Because the recertification extensions provided are considered activities by the workflow engine, any failure in those extensions is returned as a failure when the activity completes. This result causes the recertification workflow to fail, and its failure is audited in the RECERTIFICATIONLOG audit table as well.

The following extensions are provided:

- constructApprovalDocument
- recertificationMark
- recertificationMarkAccess
- recertificationSuspend
- recertificationCertify
- recertificationCertifyAccess
- recertificationAdminCertify
- recertificationAdminCertifyAccess
- remediateAccountsAndGroups
- remediateRoleMemberships
- updateRecertificationStatusAllApproved
- updateRecertificationStatusEmptyDocument

recertificationMark

The public ProcessResult recertificationMark(Account) extension updates erLastRecertificationAction for the target type, updating the erLastRecertificationAction attribute to:

```
com.ibm.itim.dataservices.model.domain.Account.REJECTED_MARK = 'REJECTED_MARK'
```

The recertification action is audited in RECERTIFICATIONLOG table for use by reports.

constructApprovalDocument

The public ProcessResult constructApprovalDocument(Person, RecertificationPolicy) extension constructs the PackagedApprovalDocument that is required for user-based recertification. This document contains all of the static roles, accounts, and groups for the specified person.

If there are no recertification targets for the person, this method returns a ProcessResult with a WARNING summary and an embedded message. In this case, it contains an output parameter list with an empty document. Otherwise, if successful, the ProcessResult contains a populated document for this particular person.

recertificationMarkAccess

The public ProcessResult recertificationMark(UserAccessAccount) extension has the same function for accesses as recertificationMark() has for users and accounts. It updates the erAccessLastRecertificationAction attribute specific to the UserAccess passed in to:

```
com.ibm.itim.dataservices.model.domain.Account.REJECTED_MARK = 'REJECTED_MARK'
```

The recertification action is audited in RECERTIFICATIONLOG table for use by reports.

Note: This method is for suspending accounts only. No method for suspending access is provided.

recertificationSuspend

The `public ProcessResult recertificationSuspend(Account)` extension updates `erLastRecertificationAction` for the account. It updates the `erLastRecertificationAction` attribute to:

```
com.ibm.itim.dataservices.model.domain.Account.REJECTED_SUSPEND = 'REJECTED_SUSPEND'
```

The recertification action is audited in `RECERTIFICATIONLOG` table for use by reports.

Note: This method is for suspending accounts only. No method for suspending access is provided.

recertificationCertify

The `public ProcessResult recertificationCertify(Account) extension` updates `erLastRecertificationAction` for the target type. It updates the `erLastRecertificationAction` attribute to:

```
com.ibm.itim.dataservices.model.domain.Account.CERTIFIED = 'CERTIFIED'
```

The recertification action is audited in `RECERTIFICATIONLOG` table for use by reports. This extension also updates the `erLastCertifiedDate` attribute with the current timestamp.

recertificationCertifyAccess

The `public ProcessResult recertificationCertify(UserAccessAccount)` extension updates `erLastAccessRecertificationAction` for the access. It updates the `erLastRecertificationAction` attribute for the specified `UserAccess` to:

```
com.ibm.itim.dataservices.model.domain.Account.CERTIFIED = 'CERTIFIED'
```

The recertification action is audited in `RECERTIFICATIONLOG` table for use by reports. This extension also updates the `erAccessLastCertifiedDate` attribute for the `accessAttribute` with the current timestamp.

Note: This method is the access version of `recertificationCertify` for users and accounts.

recertificationAdminCertify

The `public ProcessResult recertificationAdminCertify(Account)` extension updates `erLastRecertificationAction` for the target type. It updates the `erLastRecertificationAction` attribute to:

```
com.ibm.itim.dataservices.model.domain.Account.CERTIFIED_ADMIN = 'CERTIFIED_ADMIN'
```

The recertification action is audited in `RECERTIFICATIONLOG` table for use by reports. This extension also updates the `erLastCertifiedDate` attribute with the current timestamp.

recertificationAdminCertifyAccess

The `public ProcessResult recertificationAdminCertify(UserAccessAccount)` extension updates `erLastRecertificationAction` for the access. It updates the `erAccessLastRecertificationAction` attribute for the `UserAccess` passed in to:
`com.ibm.itim.dataservices.model.domain.Account.CERTIFIED_ADMIN = 'CERTIFIED_ADMIN'`

The recertification action is audited in `RECERTIFICATIONLOG` table for use by reports. This extension also updates the `erAccessLastCertifiedDate` attribute for the `accessAttribute` with the current timestamp.

Note: This method is the access version of `recertificationAdminCertify` for users and accounts.

remediateAccountsAndGroups

The public `ProcessResult remediateAccountsAndGroups(PackagedApprovalDocument, Person, RecertificationPolicy, String)` extension runs user recertification remediation on all of the accounts, groups, and accesses in the approval document. Each entry is processed based on the responses in the document and the enforcement action of the policy. Any recertification status updates are performed directly through data services. Any removals of accounts, groups, or accesses are handled by launching the appropriate workflow operation as a subprocess.

remediateRoleMemberships

The public `ProcessResult remediateRoleMemberships(PackagedApprovalDocument, Person, RecertificationPolicy, String)` extension runs user recertification remediation on all role memberships in the approval document. Each entry is processed based on the responses in the document and the enforcement action of the policy. If any roles are removed, this extension launches the person modify operation to process the removal and corresponding policy enforcement actions. If no role are removed, this activity directly invokes policy enforcement to make sure that the recertification is performed on every person.

updateRecertificationStatusAllApproved

The public `ProcessResult updateRecertificationStatusAllApproved(PackagedApprovalDocument, Person, RecertificationPolicy)` extension processes the approval document, and updates the recertification status of each entry. The entries include accounts, groups, and role memberships. This extension is only invoked when all choices in the document are approved. Different extensions are used for remediation. Any recertification status updates are performed directly through data services.

updateRecertificationStatusEmptyDocument

The public `ProcessResult updateRecertificationStatusEmptyDocument(PackagedApprovalDocument, Person, RecertificationPolicy)` extension updates the required recertification status on the person being recertified. It is the only action required in the case that the document does not contain any resources. The recertification status updates are performed directly through data services.

Wait extension

The wait extension pauses the workflow until a specified time.

Overview

A wait extension is code that can be called directly from a workflow. It is implemented in the `WaitExtension` class in the `com.ibm.itim.workflowextensions` package.

The following extension is provided:

- `scheduleTimeout`

scheduleTimeout

The `public ProcessResult scheduleTimeout(Date)` extension suspends the workflow until the time specified by `Date`, which is the standard Date object in JavaScript. When the specified time is reached, the extension activity is complete and the workflow continues.

Embed the wait extension in a loop in the workflow if you want the workflow to check a condition and continue only when the condition is no longer met. The loop requires the following logic:

- Check the condition.
- Calculate the target date for the wait extension from the current date. Use JavaScript.
- Run the wait extension. Use the calculated target date for `scheduleTimeout(Date)`.

For more information about `Date`, see a JavaScript reference like the following: JavaScript Date Reference. Another possible reference is the ECMAScript(r) Language Specification, published by ECMA International, which now administers the standards that are the basis for JavaScript and other scripting languages.

Examples

- A workflow loop checks CPU load and continues only when CPU load falls below the desired level.
  1. Check CPU load.
     - If CPU load is below the desired threshold: Exit the loop.
     - If CPU load is above the desired threshold: Calculate the target `DATE` and then run the wait extension.
  2. When the wait extension is complete, loop to check CPU load again.
- Enforce dynamically calculated timeouts for long-running workflow activities. For example, implement an approval that is pending for *two working days*.
  1. Calculate the target `DATE`. Use JavaScript. The calculation needs to account for workflows that are triggered near a weekend. For example, consider the desired period of two working days. If the workflow is triggered on a Friday, the target date is Tuesday (four elapsed days). If the workflow is triggered on a Monday, then the target date is Wednesday (two elapsed days).
  2. Branch workflow execution that uses a fork type of `AND`. Put the approval on one branch and the wait extension with the target `DATE` on the other branch.
  3. Merge the two branches with a join type of `OR`.

The workflow continues when either branch is complete: an approval is submitted or the wait extension times out.

Chapter 7. REST APIs

You can develop custom applications by using the REST application programming interfaces (APIs) that come with the IBM Security Identity Manager. The REST APIs are available so that you can administer the tasks outside of the IBM Security Identity Manager user interface. The topic provides information about the functions that REST APIs support.

The REST APIs are segregated into a set of functional components of IBM Security Identity Manager that are listed in the following section.

Person Management
View or edit user profiles.

System User Management
Search capability for the IBM Security Identity Manager system users based on unique identifiers.

Password Management
Change or reset the password, and recover the forgotten password.

Access Management
Request, view, edit, or delete the access.

Activity Management
View and act on your activities.

Delegation Management
Delegate activities, view, edit, and delete the delegation schedule.

Generic Search APIs
Assorted set of search capabilities that are provided by the REST APIs.

In addition, IBM Security Identity Manager virtual appliance provides a set of APIs for the following administration or management tasks.

Widgets
View cluster status, server controls, statistics, notifications, middleware and server monitoring, and other information.

Analysis and Diagnostics Monitoring
Analysis and diagnostics tools such as memory statistics, CPU usage, performance metrics, service statistics, monitoring, and other aspects.

System Settings Management
Control of system settings such as host name, date or time, network settings, audit events, support files, snapshots, SNMP monitoring, licensing, firmware settings, fix packs, and other aspects.

Management and Application Interfaces Management
View and work on your management and application interfaces.

Hosts File management
Handle host names, IP addresses, and other services.

Access the REST API documentation

REST APIs are packaged as compressed files.

1. Log on to the IBM Security Identity Manager virtual appliance console and open the Appliance Dashboard page.
2. From the top-level menu of the **Appliance Dashboard**, select **Configure** > **Advanced Configuration** > **Custom File Management**.
3. In the **All Files** tab, browse to **directories** > **utilities**.
4. Select and download each of the following files:
   - `RAPI_DOCS.zip`
   - `extensions.zip`
5. Extract the compressed files to a local folder.
6. Open the `index.html` page to view the following REST API documentation:

   For IBM Security Identity Manager virtual appliance-related REST APIs
   Open `Virtual_Appliance_REST_API_Docs/index.html`.

   For IBM Security Identity Manager related REST APIs
   Open `extensions/7.0/doc/rest/index.html`.

REST API code samples

The REST API code samples are annotated. The annotations provide information about how to use the samples in your test environment.

The REST API annotated code samples are available in the `extensions/7.0/examples` directory of the `extensions.zip` file.

Do these steps to access the REST API annotated code samples:

1. Log on to the IBM Security Identity Manager virtual appliance console to open the **Appliance Dashboard**.
2. From the top-level menu of the **Appliance Dashboard**, select **Configure** > **Advanced Configuration** > **Custom File Management** to display the Custom File Management page.
3. Click the **All Files** tab.
4. Go to `directories/utilities`.
5. Select `extensions.zip` and click **Download**.
6. Extract the `extensions.zip` file.
7. Go to `extensions/7.0/examples`.

Invoking REST APIs in a domain different from the originating web page

IBM Security Identity Manager REST APIs support cross-origin resource sharing (CORS). CORS describes a mechanism for supporting requests that a web page sends to a server that is not in the same domain as the originating web page. You can configure CORS to control which origins can work with the IBM Security Identity Manager REST APIs.

About this task

You can modify a list of trusted domains that can access Identity Service Center REST APIs. Complete the steps.

Procedure

1. From the top-level menu of the **Appliance Dashboard**, select **Configure** > **Advanced Configuration** > **Update Property**.
2. On the Update Property page, click the **All Properties** tab.
3. Click the **Identity server property files** tab.
4. Select the `rest.properties` file to work with it.
5. Click **New** to add a property as `ui.CORSOrigin`. For more information, see Managing the server properties.
6. In the `ui.CORSOrigin` property, set the trusted domains. You can add multiple domains that are separated by white space.

Results

The domains that are listed in the `ui.CORSOrigin` property can only access the IBM Security Identity Manager REST APIs.

Filter configuration for REST search services

Use the following information to learn how the IBM Security Identity Manager REST search services create the search filter expression.

You can configure the filters and the HTTP request URL query parameters to control the data that the REST search services return.

Note:

To use a specific filter configuration for a request, the REST client can supply `'filterId'` as a URL query parameter and its value must be the filter identifier that is configured in the `searchfilter.json` file. See "Examples" on page 32.

For more information about how to define the filter identifier, see Defining the filter identifier for REST search service. The REST service uses the corresponding filter configuration in the following table to create the filter expression.

Table 1. Filters and their supported values

Filters	Description
`"filterTemplate"`	A template string for the filter expression. For example, • `"(&(&(date>=${fromDate})(date<=${toDate}))${filterExpression})"`. *fromDate* and *toDate* are the URL parameter names and their values are placed in the template. • `${filterExpression}` is replaced with the expression that is created with remaining URL parameters as described in the table. **Note:** `filterTemplate` is an optional configuration for a filter. If the `filterTemplate` is not specified, then it is equivalent to `"filterTemplate": "${filterExpression}"`.
`"joinOperator"`	An operator that is applied to join the logical expressions. Supported values are **&** and **|**.
`"multivalueJoinOperator"`	An operator that is applied to join the logical expressions that are created for the multiple value URL parameters. Supported values are **&** and **|**.
`"comparisonOperator"`	An operator that is applied for an attribute and its value comparison. Supported values are **=**, **!=**, **~=**, **>=**, **<=**, **>**, **<**.

Table 1. Filters and their supported values (continued)

Filters	Description
`"baseFilter"`	You can substitute attributes of the current Identity Service Center account or the owner of the account into the base search filter. These attributes are used when the filter is evaluated. The notation ${*xxxx*} is used to specify where the substitution is made, and *xxxx* specifies what attribute value is to be substituted. The special string **systemUser** represents the user account of the current Identity Service Center user. You can qualify **systemUser** to specify an account attribute, such as **${systemUser.eruid}**. You can also reference attributes of the owner of the account, such as **${systemUser.owner.cn}**. Only attributes of the current account or the owner of the account can be used as substitutions into the base search filter. If a substitution cannot be evaluated or is evaluated to an empty string, a substitution value of `_undefined_` is used instead. For example, `"baseFilter": "(!(uid=${systemUser.owner.uid}))"`
`"allowWildcard"`	Specifies whether to use `*` as wildcard in the final filter expression or escape it. Supported values are **true** and **false**.

Rules that apply to populate the `filterTemplate`

- If a parameter in the template is not supplied as URL query parameter in the HTTP request, it is removed from the expression. For example,

  The `filterTemplate` is `"(&(cn=xyz)(sn=${sn}))"` and

  the request URL is `"/rest/people"`

  The resultant expression is `(cn=xyz)`.
- String `${filterExpression}` in the `filterTemplate` is replaced by the filter expression that is created by using the filter configuration and URL parameters that are not provided in the filter template. For example,

  The `filterTemplate` is `"(&(cn=xyz)(sn=${sn})${filterExpression})"` and

  the request URL is `"/rest/people?sn=abc&email=pqr@site.com"`

  The resultant expression is `(&(cn=xyz)(sn=abc)(email=pqr@site.com))`. In this example, `sn`, `email` are two URL query parameters but `email` is used to create `$filterExpression` because `sn` is already used in the template.

Conditions in the `filterExpression` for `joinOperator`, `multivalueJoinOperator`, `comparisonOperator`, `allowWildcard`

- If a URL parameter contains multiple values, then the template expression for that parameter is constructed by using `multivalueJoinOperator`.

  The `filterTemplate` is `"(&(cn=pqr)(sn=${sn}))"` and

  the request url is `"/rest/people?sn=abc&sn=xyz"` and

  the `multivalueJoinOperator` is `|`

  The resultant expression is `(&(cn=pqr)(|(sn=abc)(sn=xyz)))`.
- If a URL parameter contains single value, then that value is placed in the template.

  The `filterTemplate` is `(&(cn=abc)(sn=${sn}))"` and

  the request url is `"/rest/people?sn=xyz"`

  The resultant expression is `(&(cn=abc)(sn=xyz))`.

Examples

Example 1 - Person search without using the filter identifier

The PERSON_SEARCH is the REST service endpoint key for the person search capability. You must set a value for the PERSON_SEARCH that you can use as a filter identifier for person search capability when you create a request URL. You might not know the REST service endpoint keys for all the supported functions. You can use the dictionary service to know about all the supported REST service endpoint keys. Access https://*hostname*:*port*/itim/rest/dictionary to find the REST service endpoint keys.

You want to search for a person. Example 1 explains how to use the REST service, without providing any explicit filter identifier. Complete the following steps:

1. From the **Appliance Dashboard**, go to **Configure** > **Advanced Configuration** > **Update Property**.
2. On the Update Property page, click **Identity server property files** and select rest.properties.
3. Create the PERSON_SEARCH property if it does not exist. For more information, see Managing the server properties.
4. Set the value for the PERSON_SEARCH property. For example, customPersonSearch.
5. From the **Appliance Dashboard**, go to **Configure** > **Advanced Configuration** > **Custom File Management**.
6. On the Custom File Management page, click **All Files** and select directories/rest/searchfilter.json.
7. Define the customPersonSearch filter in the searchfilter.json file. For example,

```
"customPersonSearch": {
    "joinOperator": "&",
    "multivalueJoinOperator": "|",
    "comparisonOperator": "=",
    "baseFilter": "(!(uid=${systemUser.owner.uid}))",
    "allowWildcard": "false"
}
```

If the request URL is:

/itim/rest/people?cn=*abc*&sn=*pqr*&sn=*xyz**

and you log in as a user *user1*

Then, the filter expression is:

(&(&(cn=*abc*)(|(sn=*pqr*)(sn=*xyz*\2a)))(!(uid=*user1*)))

Example 2 - Request search by using the filter identifier

You want to search for the requests. Example 2 explains how to use the REST service with the filter identifier. Complete the following steps.

1. Assume that the filter identifier requestSearch is already defined for the request search REST service endpoint key.
2. Define the requestSearch filter in the searchfilter.json file. For example,

```
"requestSearch": {
    "filterTemplate":
    "(&(&(date>=${fromDate})(date<=${toDate}))${filterExpression})",
    "comparisonOperator": "=",
```

```
        "joinOperator": "|",
        "multivalueJoinOperator": "|",
        "allowWildcard": "true"
    }
```

If the request URL is:

```
/itim/rest/requests/quicksearches?filterId=requestSearch&fromDate=1425061800000
&toDate=1427826513600&accessName=*finance*&justification=*payroll*&limit=5
```

Then, the filter expression is:

```
(&(&(&(date>=1425061800000)(date<=1427826513600))(|(justification=*payroll*)(accessName=*finance*)))
```

Chapter 8. Dynamic tags in mail templates

IBM Security Identity Manager mail templates allow dynamic retrieval, substitution, and decision making in creating a message.

Dynamic content tags and examples

Security Identity Manager provides dynamic content tags to allow text substitution and enable translation. The tags are used for the emails that are generated by these tasks:

- Designing workflows
- Specifying mail activity
- Manual service notification
- Recertification notification
- Post office
- Reminder template
- Default system notifications
- Delegation notifications

These tags are associated with dynamic content:

JavaScript code
Handles JavaScript and runs the JavaScript content that is contained between the open and close tags. This tag contains child tags unless they return a string. JavaScript code is called in <JS>*MyJavaScriptCode*</JS> delimiters.

Table 2. Syntax and example of using JavaScript code to replace message content.

Syntax	Example

<JS>*text or JavaScript tag*</JS>	Enter each <JS></JS> statement as a single line:
	`An account request has been initiated for <JS>process.requesteeName;</JS>`
	`<JS>if (var x=process.getParent() !=null) return x</JS>`

<JS escapeentities="false">*text or JavaScript tag*</JS>	When specified as "false", any text that is returned by the JavaScript execution does not have its HTML entity tags escaped. For instance, the < character does not return as <. This option might be useful when the execution of the JavaScript code returns XML. For example, embedding XHTML body notifications inside the XHTML body of the post office template. The default for this attribute is "true", so not specifying the tag escapes the characters.

Table 2. Syntax and example of using JavaScript code to replace message content. (continued)

Syntax	Example
`<JS removexhtmlheader="false">text or JavaScript tag</JS>`	If `removexhtmlheader="true"` is in the `JS` tag, any text that is returned from the JavaScript does not have the DTD statement in the XHTML content. The text that is returned from the JavaScript has the DTD statement in the XHTML content when either of the following conditions exist: • `removexhtmlheader="false".` • It is not placed in the `JS` tag. The default value of this attribute is `false`. Not specifying the flag in the tag puts the DTD statement in the XHTML content.

Replace tag

Formats the message that is represented by the key to allow string replacement. The formatted string can have zero or more parameters. Parameters can contain strings, activity IDs, or JavaScript. The string inside the key must exist in the `CustomLabels.properties` file. Strings are sourced from a `CustomLabels.properties` resource bundle file or from the `Labels.properties` file.

The key of the string replacement can be specified with the key attribute or by adding a **KEY** tag between **RE** tags. Specifying a key that uses both the attribute and tag at the same time results in an exception.

The tag has these parameters:

Key Represents the resource bundle key for a **RE** tag. For example:

```
<RE key="key">
</RE>
```

PARM Represents the parameters for a **RE** tag. For example:

```
<RE key="key">
<PARM>with plain text</PARM>
</RE>
```

Table 3. Syntax and examples of using a RE tag to replace message content.

Syntax	Example

`<RE key="message key">`
`<PARM>`*`text or JavaScript tag`*`</PARM>`
`</RE>`
 	
or enter each `<KEY></KEY>` statement as a single line:
`<RE><KEY>`*`message key or JavaScript tag to return a key`*
`</KEY>`
`<PARM>`*`text or JavaScript tag`*`</PARM>`
`</RE>`
 	
The `KEY` can be specified by either an attribute on the `RE` tag, or as a subelement of the `RE` tag by using the tag `KEY`.
	`<RE key="message key">`
	`<PARM>with plain text</PARM>`
	`<PARM><JS>process.requesteeName;`
	`</JS></PARM></RE>`
	
	Output:
	`This is a formatted string replacement example` **`with plain text`** `and JavaScript code for requestee name` **`John Smith.`**

Table 3. Syntax and examples of using a RE tag to replace message content. (continued)

Syntax	Example

To enable string replacement for translation, specify a custom label in a `CustomLabels.properties` file to overwrite a `Labels.properties` key.
 	
For example, the `Labels.properties` file contains this key/value pair.
`readOnlyDateFormat=MMM dd, yyy hh:mm:ss z`
 	
To override this format, add the same key to the `CustomLabels.properties` file.
	`<RE key="readOnlyDateFormat">`
	`<PARM><JS>if (process.scheduled !=null)`
	`return process.scheduled.getTime();`
	`else`
	`return "";</JS></PARM></RE>`
	
	Output:
	`Apr 18, 2005 05:20:52 EDT`

Non-compliant message tag

Represents a message that describes the noncompliant attributes of an account. For example:

```
<CAMessage/>
```

Dynamic content message tags

Tags are delimited in `<TAG/>` syntax, such as the following examples:

Table 4. Syntax and example of using tags to replace message content.

Syntax	Example
`<TagName/>`	`<CAMessage/>` Returns a string that describes the non-compliant attributes of an account.
	`<ManualServiceAddAccount/>` Returns a string that contains the text body for manual service email notification.
	`<rfiActivityHasBeenSubmitted/>` Returns a string that contains the text body of an RFI activity that was submitted in an account request workflow.

ID tag Represents the activity ID in the form: `Process.ActivityId.` For example:

```
<ID/>
```

ITIMURL tag

Based on group membership of the person. It represents the URL of the IBM Security Identity Manager Server. A forced URL can be applied by using the `forcedurl` attribute of the tag. This attribute contains constant values such as the value `console`, `enduser`, or `ISC`.

Table 5. Syntax and examples of ITIMURL.

Syntax	Example
`<ITIMURL/>`	Based on group membership of the person. It represents the URL of the IBM Security Identity Manager Server.

Table 5. Syntax and examples of ITIMURL. (continued)

Syntax	Example
`<ITIMURL forcedurl="enduser"/>`	Represents the URL of the graphical user interface on the IBM Security Identity Manager Server. If the `forcedurl` attribute is used, the URL is not generated based on the group membership of the person. These values are associated with this attribute: **`enduser`** The URL points at the self-service graphical user interface. **`console`** The URL points at the administrator graphical user interface. **`servicecenter`** The URL points at the service center graphical user interface.
`<ITIMURL forcedurl="console"/>`	
`<ITIMURL forcedurl="servicecenter"/>`	

Properties file values

To change templates, you can add a property in the `CustomLabels.properties` file or create your own properties and values by using the Update Property page from the **Appliance Dashboard** of the IBM Security Identity Manager virtual appliance console. See Managing the server properties.

Required escape characters and JavaScript

The following characters must be escaped by using the appropriate HTML entity form that has the format `&entity;`. This action ensures that the notification template XML is well-formed.

Table 6. Escape characters

Escape character	Character
`<`	Less Than (<)
`>`	Greater Than (>)
`&quote;`	Quotation (")
`&apos;`	Apostrophe (')
`&`	Ampersand (&)

For example, to use the following JavaScript

```
if (i<4) return "less than four";
```

the dynamic content tag is

```
<JS> if (i&lt;4) return &quote;less than four&quote;;</JS>
```

Common formatting patterns in the XHTML body

Default messages are formatted with a common pattern in the XHTML body and also contain message-unique statements.

For example, the XHTML for the to-do reminder template calls a common style sheet (the imperatives.css file) and logos. Message-unique statements are similar to the following ones:

```
<!-- Start of notification body -->
      <textBody/>
        <RE key="escalation_note"/> <escalationTime/>
     </td>
    </tr>
<!-- End of notification body -->
```

The following example shows a complete set of statements in an XHTML body:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html
     PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
<title>$TITLE</title>
<meta content="text/html; charset=UTF-8" http-equiv="Content-Type" />
<link type="text/css" title="Styles" rel="stylesheet"
 href="$BASE_URL/console/css/imperative.css" />
</head>

<!-- Put Next statement on one line -->

<body topmargin="0" marginheight="0" leftmargin="0" marginwidth="0"
 bgcolor="ffffff">

<!-- Block for the Template Header part  -->
<table width="100%" border="0" cellpadding="0" cellspacing="0">
 <tbody>
  <tr>
   <!-- Security logo -->
   <td width="186" background="$BASE_URL/console/html/images/mid-part-1.gif">
   <img src="$BASE_URL/console/html/images/left-tiv-1.gif" alt="$LOGO_ALT" /></td>
   <!-- Middle part -->
   <td background="$BASE_URL/console/html/images/mid-part-1.gif" width="692"></td>
   <!-- IBM logo -->
   <td background="$BASE_URL/console/html/images/ibm_banner.gif" width="96"></td>
  </tr>
 </tbody>
</table>

<!-- Title Bar -->
<table width="100%" border="0" cellpadding="0" cellspacing="0">
 <tbody>

  <tr>
   <td background="$BASE_URL/console/html/images/titlebar_middle.gif"
   height="23" width="8">
   <img border="0" src="$BASE_URL/console/html/images/titlebar_left.gif"
    width="10" height="23" /></td>
   <!-- ISIM Notification Lable -->
<td background="$BASE_URL/console/html/images/titlebar_middle.gif"
  height="23" classpath="portfolio-header" width="979">$TITLE</td>
   <td background="$BASE_URL/console/html/images/titlebar_middle.gif"
   height="23" width="5"><img
    border="0" src="$BASE_URL/console/html/images/titlebar_right.gif"
    width="5" height="23" /></td>
  </tr>
 </tbody>
</table>

<table width="100%" border="0" cellpadding="0" cellspacing="0">
 <tbody>
```

```
  <tr>
   <!-- Backgroud for the template body -->
   <td background="$BASE_URL/console/html/images/portfolio_background.gif"
   height="148">
   <table border="0" cellspacing="0" cellpadding="0" width="100%">
                <tr>
                    <td align="left" class="text-description" height="65">
                    <!-- Start of notification body -->
      <textBody/>
        <RE key="escalation_note"/> <escalationTime/>
     </td>
    </tr>
        <!-- End of notification body -->           </table>
     </td>
  </tr>
 </tbody>
</table>
<!-- Copy Right Table -->
<table width="100%" border="0" cellpadding="0" cellspacing="0">
 <tbody>
  <tr bgcolor="#9d9d9d" align="center" valign="middle">
   <td class="text-description"><span class="cont1" id="W57ea57ea"><span
    class="txt" id="text">IBM Copyright 2007</span></span></td>
  </tr>
 </tbody>
</table>
</body>
</html>
```

Mail templates

You define mail templates to deliver customized message notifications. The templates use several customization functions.

Templates have these main parts:

Subject
: Describes an activity to a recipient of the notification. The subject can consist of plain text and dynamic content tags. If no subject is specified for manual service activities, no email is sent.

Text body
: Describes the outcome of an activity, such as an account approval. The content can consist of plain text, dynamic content tags, and JavaScript code.

XHTML body
: Provides the content of the email as an HTML message.

Dynamic content can include dynamic content message tags, JavaScript code, and tags that replace variables with other values or reference a property that allows translation with the `CustomLabels.properties` file.

Manual service default messages

IBM Security Identity Manager provides default notification templates for messages that participants that are service owners receive when changes occur to accounts or passwords for manual services that they own.

Default notification templates

IBM Security Identity Manager provides these default notification templates:

<ManualServiceAddAccount/>
Provides default text sent to a participant when an account is added for the user of a manual service.

<ManualServiceModifyAccount/>
Provides default text sent to a participant when an account is modified for the user of a manual service.

<ManualServiceDeleteAccount/>
Provides default text sent to a participant when an account is deleted for the user of a manual service.

<ManualServiceRestoreAccount/>
Provides default text sent to a participant when an account is restored for the user of a manual service.

<ManualServiceSuspendAccount/>
Provides default text sent to a participant when an account is suspended for the user of a manual service.

<ManualServiceChangePassword/>
Provides default text sent to a participant when a password change occurs for the user of a manual service.

Properties used for translation

If the properties exist in the `CustomLabels.properties` file, their value is used. Otherwise, the values of the properties in `Labels.properties` file are used. These properties contain the translated versions of the messages (with parameter substitution) that make up the dynamic tags. Change their values in the `CustomLabels.properties` file if you want different text. Do not change the defaults in the `Labels.properties` file.

The properties include these items:

```
manualServiceWorkOrderAddOperationMessage
manualServiceAttributeName
manualServiceAttributeValue
manualServiceAttributeAction
manualServiceAddAction
manualServiceRemoveAction
manualServiceReplaceAction
manualServiceWorkOrderChangePwdOperationMessage
manualServiceWorkOrderPwdValueMessage
manualServiceWorkOrderDeleteOperationMessage
manualServiceWorkOrderModifyOperationMessage
manualServiceWorkOrderRestoreOperationMessage
manualServiceWorkOrderSuspendOperationMessage
manualServiceUnknownPerson
```

Notification script example

A default notification script for a manual service provides a message that is sent to a participant. For example, the `ManualServiceAddAccount` notification output is similar to this example:

```
Attribute Name: Attribute Value
myattr: TT
Password: secret
Owner: Auditor
User ID: auditor1
```

```
Description: manual service operation
Requestee: Auditor
Subject: auditor1
Request Initiated: Jun 28, 2007 05:11:05 IST

  Requested by process:
  Process Name: Account Add
  Description: Account Add Process
  Requester: System Administrator
  Requestee: Auditor
  Subject: auditor1
```

Output example

The <ManualServiceAddAccount/> template provides a message that uses some of the values in the Labels.properties file:

```
manualServiceWorkOrderAddOperationMessage
manualServiceAttributeName : manualServiceAttributeValue
{insert real attribute names here} : {insert real attribute values here}
```

The <ManualServiceModifyAccount/> tag generates:

```
manualServiceWorkOrderModifyOperationMessage
{Place the following attributes on one line:}
manualServiceAttributeName : manualServiceAttributeValue
   : manualServiceAttributeAction
{insert real attribute names here} : {insert real attribute values here} :
{depending on what needs to be done, one of the following: }
{Place the following attributes on one line:}
{manualServiceAddAction,manualServiceReplaceAction,
   manualServiceRemoveAction}
```

Recertification default messages

IBM Security Identity Manager provides default message templates for recertification messages.

Default recertification templates

IBM Security Identity Manager provides default message templates for recertification messages. You cannot change the following templates:

Suspend Account

Provides default text that requests a participant to recertify use of an account. Declining the request suspends the account.

For example, the participant receives this message:

```
Recertification required for account myaccount on service shortword-linux

 You have received a recertificaton request for account myaccount on service
 shortword-linux owned by firstname lastname.

 Rejection of this recertification request will result in the suspension of
 account myaccount on shortword-linux.

 Activity:Recertification of Account/Access
 Date submitted:Apr 26, 2007 10:34:51 IST
 Request type:Recertification
 Requested for:firstname lastname
 Requested by:SYSTEM
```

```
Access/Account:myaccount
Description:

Due date:Apr 27, 2007 10:34:57 IST
```

Delete Account

Provides default text that requests a participant to recertify use of an account. Declining the request deletes the account.

For example, the participant receives this message:

```
Recertification required for account myaccount on service shortword-linux

 You have received a recertificaton request for account myaccount
 on service shortword-linux owned by firstname lastname.

 Rejection of this recertification request will result in the deletion
 of account myaccount on shortword-linux.

 Activity:Recertification of Account/Access
 Date submitted:Apr 26, 2007 10:34:51 IST
 Request type:Recertification
 Requested for:firstname lastname
 Requested by:SYSTEM
 Access/Account:myaccount
 Description:
 Due date:Apr 27, 2007 10:34:57 IST
```

Mark Account

Provides default text that is sent to a participant to recertify use of an account. Declining the request marks the account for a subsequent action on the account.

For example, the participant receives this message:

```
Recertification required for account myaccount on
 service shortword-linux.

 You have received a recertificaton request for account myaccount on service
 shortword-linux owned by firstname lastname.

 Rejection of this recertification request will result in
 account myaccount on shortword-linux being marked as rejected
 for recertification.

 Activity:Recertification of Account/Access
 Date submitted:Apr 26, 2007 10:34:51 IST
 Request type:Recertification
 Requested for:firstname lastname
 Requested by:SYSTEM
 Access/Account:myaccount
 Description:
 Due date:Apr 27, 2007 10:34:57 IST
```

Mark Access

Provides default text that is sent to a participant to recertify use of an account on an access. Declining the request marks the access for a subsequent action on the account.

For example, the participant receives this message:

```
Recertification required for account myaccount on access myaccess.

 You have received a recertificaton request for account myaccount on
 access myaccess owned by firstname lastname.

 Rejection of this recertification request will result in
 access myaccess being marked as rejected for recertification.
```

```
Activity:
Date submitted:Apr 26, 2007 10:34:51 IST     /* Need to fill this data */
Request type:
Requested for:
Requested by:
Access/Account:
Description:
Due date:Apr 27, 2007 10:34:57 IST
```

Delete Access

Provides default text that requests a participant to recertify use of an account on an access. Declining the request deletes the account on the access.

For example, the participant receives this message:

```
Recertification required for account myaccount on access myaccess.

 You have received a recertificaton request for account myaccount on
 access myaccess owned by firstname lastname.

 Rejection of this recertification request will result in the deletion
 of access myaccess.

 Activity:
 Date submitted:Apr 26, 2007 10:34:51 IST    /* Need to fill this data */
 Request type:
 Requested for:
 Requested by:
 Access/Account:
 Description:
 Due date:Apr 27, 2007 10:34:57 IST
```

Account Suspended

Provides default text that is sent to a participant, confirming suspension of an account, after a participant declines a recertification request.

For example, the participant receives this message:

```
Account myaccount on service shortword-linux has been suspended due
  to rejection  of a recertification request

 The account myaccount on service shortword-linux owned by
 firstname lastname has been suspended due to rejection of
 a recertification request.

 Activity:Recertification of Account/Access
 Date submitted:Apr 26, 2007 10:34:51 IST
 Request type:Recertification
 Requested for:firstname lastname
 Requested by:SYSTEM
 Access/Account:myaccount
 Description:
```

Account Deleted

Provides default text that is sent to a participant, confirming deletion of an account, after a participant declines a recertification request.

For example, the participant receives this message:

```
Account myaccount on service shortword-linux has been deleted due to
 rejection of a recertification request

 The account myaccount on service shortword-linux owned by
 firstname lastname has been deleted due to rejection
 of a recertification request.

 Activity:Recertification of Account/Access
 Date submitted:Apr 26, 2007 10:34:51 IST
```

```
Request type:Recertification
Requested for:firstname lastname
Requested by:SYSTEM
Access/Account:myaccount
Description:
Due date:Apr 27, 2007 10:34:57 IST
```

Account Marked

Provides default text that is sent to a participant, confirming that an account is marked for suspension, after a participant declines a recertification request.

For example, the participant receives this message:

```
Account myaccount on service shrotword-linux has been marked as rejected
 for recertification due to rejection of a recertification request

 The account myaccount on service shortword-linux owned by
 firstname lastname has been marked as rejected for recertification
 due to rejection of a recertification request.

 Activity:Recertification of Account/Access
 Date submitted:Apr 26, 2007 10:34:51 IST
 Request type:Recertification
 Requested for:firstname lastname
 Requested by:SYSTEM
 Access/Account:myaccount
 Description:
 Due date:Apr 27, 2007 10:34:57 IS
```

Access Marked

Provides default text that is sent to a participant. It confirms that an account on an access is marked for subsequent action after a participant declines a recertification request.

The template contains these statements:

```
Account myaccount on access myaccess has been deleted
due to rejection of a recertification request.

The account myaccount on access myaccess owned by firstname lastname
has been marked as rejected for recertification due to rejection of
a recertification request.

 Activity:
 Date submitted:Apr 26, 2007 10:34:51 IST
 Request type:
 Requested for:
 Requested by:
 Access/Account:
 Description:
 Due date:Apr 27, 2007 10:34:57 IST
```

Access Removed

Provides default text that is sent to a participant, confirming deletion of an account on an access, after a participant declines a recertification request.

For example, the participant receives this message:

```
Account myaccount on access myaccess has been deleted due to
rejection of a recertification request.

The account myaccount on access myaccess owned by firsname
lastname has been deleted due to rejection of a recertification request.

 Activity:
 Date submitted:Apr 26, 2007 10:34:51 IST
 Request type:
 Requested for:
```

```
 Requested by:
 Access/Account:
 Description:
 Due date:Apr 27, 2007 10:34:57 IST
```

User Recertification Pending

Provides default text that is sent to a participant, confirming that a user recertification is pending, after a recertification request is initiated.

For example, the participant receives this message:

```
You have received a recertification request for user firstname lastname.
The recertification includes their membership in X role(s) and ownership
of Y account(s). Please indicate whether the user still requires these
resources:

The account myaccount on access myaccess owned by firsname
lastname has been deleted due to rejection of a recertification request.

 Activity: Recertification of Account/Access/User
 Date submitted: Sep 08, 2008 04:10:32 EDT
 Request type: Recertification
 Requested for: firstname lastname
 Requested by: System
 Due date: Sep 18, 2008 04:10:34 EDT
```

User Recertification Rejected

Provides default text that is sent to a participant, confirming that one or more resources were declined in a user recertification request.

For example, the participant receives this message:

```
One or more resources for user firstname lastname have been rejected
during recertification.

The account myaccount on access myaccess owned by firsname
lastname has been deleted due to rejection of a recertification request.

 Activity: Recertification of Account/Access/User
 Date submitted: Sep 08, 2008 06:30:07 EDT
 Request type: Recertification
 Requested for: firstname lastname
 Requested by: System

The following roles were rejected:
 rolename

The following accounts were rejected, along with all groups associated
with the accounts:
 Account "uid" on service "servicename"

The following groups were rejected, but the account was accepted:
 Group "groupname" for account "uid" on service "servicename"
```

Properties file values

To change templates, you can use all of the *key*=`value` statements in the `CustomLabels.properties` file, or create your own properties and values. Use the Update Property page from the **Appliance Dashboard** of the IBM Security Identity Manager virtual appliance console. See Managing the server properties.

The properties include these items on one line:

```
recertOn={0} on {1}
recertTemplateSubject=Recertification required
 for account {0} on service {1}
recertTemplateAccessSubject=Recertification required
```

```
 for account {0} on access {1}
recertTemplateBody=You have received a recertificaton request
 for account {0} on service {1} owned by {2}.
recertTemplateAccessBody=You have received a recertificaton request
 for account {0} on access {1} owned by {2}.
recertDeclineSuspendsBody=Rejection of this recertification request
 will result in the suspension of account {0} on {1}.
recertDeclineDeletesBody=Rejection of this recertification request
 will result in the deletion of account {0} on {1}.
recertDeclineMarksBody=Rejection of this recertification request
 will result in account {0} on {1} being marked as rejected for recertification.
recertDeclineDeletesAccessBody=Rejection of this recertification request
 will result in the deletion of access {0}.
recertDeclineMarksAccessBody=Rejection of this recertification request
 will result in access {0} being marked as rejected for recertification.
recertDeclinedAcctSuspendedSubj=Account {0} on service {1} has
 been suspended due to rejection of a recertification request
recertDeclinedAcctDeletedSubj=Account {0} on service {1} has
 been deleted due to rejection of a recertification request
recertDeclinedAcctMarkedSubj=Account {0} on service {1} has
 been marked as rejected for recertification due to rejection
 of a recertification request
recertDeclinedAccessDeletedSubj=Account {0} on access {1} has
 been deleted due to rejection of a recertification request
recertDeclinedAccessMarkedSubj=Account {0} on access {1} has
 been marked as rejected for recertification due to rejection
 of a recertification request
recertDeclinedAcctSuspendedBody=The account {0} on
 service {1} owned by {2} has been suspended due to rejection
 of a recertification request.
recertDeclinedAcctDeletedBody=The account {0} on service {1}
 owned by {2} has been deleted due to rejection of a recertification request.
recertDeclinedAcctMarkedBody=The account {0} on service {1}
 owned by {2} has been marked as rejected for recertification
 due to rejection of a recertification request.
recertDeclinedAccessDeletedBody=The account {0} on access {1}
 owned by {2} has been deleted due to rejection of a recertification request.
recertDeclinedAccessMarkedBody=The account {0} on access {1}
 owned by {2} has been marked as rejected for recertification
 due to rejection of a recertification request.
userRecertTemplateSubject=Recertification required for user {0}
userRecertTemplateBody=You have received a recertificaton request
  for user {0}. The recertification includes their membership in {1} role(s)
  and ownership of {2} account(s). Please indicate whether the user still
  requires these resources.
userRecertDeclinedSubj=Recertification request rejected for user {0}
userRecertDeclinedBody=One or more resources for user {0} have been
  rejected during recertification.
userRecertRolesRejectedLabel=The following roles were rejected:
userRecertAccountsRejectedLabel=The following accounts were rejected,
  along with all groups associated with the accounts:
userRecertGroupsRejectedLabel=The following groups were rejected,
  but the account was accepted:
userRecertAcctLabel=Account "{0}" on service "{1}"
userRecertGroupLabel=Group "{0}" for account "{1}" on service "{2}"
```

Recertification template key definitions

Recertification templates use the following key definitions:

```
name=Activity
timeScheduled=Date submitted
recertRequestType=Request type
recertRequestedFor=Requested for
recertRequestedBy=Requested by
recertAccountAccess=Access/Account
```

```
recertDueDate=Due date
recertRequestTypeName=Recertification
readOnlyDateFormat=MMM dd, yyyy hh:mm:ss z
```

Workflow default messages

IBM Security Identity Manager provides default workflow messages.

Default workflow templates

All the workflow notice templates can be customized. IBM Security Identity Manager provides these default workflow notice templates:

Activity Timeout Template

Provides information that the workflow activity is timed out and terminated. By default, this template is enabled.

For example, the template provides this message:

```
Workflow activity is being timed out and will be terminated
by the workflow system.

The following activity has timed out.The activity will be terminated
by the workflow system and the result set to Terminated.

Activity Information

View Changes: http://localhost:9090/itim/console
Activity ID: ADApproval
Activity: AD Account Approval
Time Started: Jun 09, 2007 12:28:45 IST
Time Completed:
Result Summary: Escalated
State: Running
Activity Type: Manual Approval/Reject

Process Information

 Process ID: 1099575082113388748
Activity: Default AD Account Approval Workflow
Description:
State:Running
Date submitted: Jun 09, 2007 12:23:41 IST
Time Completed:
Result Summary:
Requester: 1099572462907357646
Requestee: firstname lastname
Subject:
Comment:
Detail:
```

The subject statement is:

```
<RE key="activity_timeout_subject" />
```

The plain text is:

```
<RE key="activity_timeout_message" />

<RE key="activity_timeout_detail" />

<RE key="activityInformation" />
<ITIMURL/>
<RE key="activityID"/>: <JS>activity.id;</JS>
<RE key="name"/>: <JS>activity.name;</JS>
<RE key="timeStarted"/>: <RE key="readOnlyDateFormat"><PARM>
 <JS>if (activity.started != null)
 return activity.started.getTime();
 else return '';</JS></PARM></RE>
 <RE key="timeCompleted"/>: <RE key="readOnlyDateFormat"><PARM>
  <JS>if (activity.completed != null)
  return activity.completed.getTime();
  else return '';</JS></PARM></RE>
```

```
<RE key="resultSummary"/>: <RE><KEY>
 <JS>process.STATE_PREFIX + activity.resultSummary;
 </JS></KEY></RE>
<RE key="state"/>: <RE><KEY><JS>process.STATE_PREFIX+activity.state;
 </JS></KEY></RE>
<RE key="activityType"/>: <RE><KEY>
 <JS>activity.TYPE_PREFIX + activity.type;</JS>
 </KEY></RE>
<RE><KEY><JS>activity.TYPE_PREFIX + activity.subtype;</JS></KEY></RE>

<RE key="processInformation" />

<RE key="processID"/>: <JS>process.id;</JS>
<RE key="name"/>: <RE><KEY><JS>process.name;</JS></KEY></RE>
<RE key="description"/>: <RE><KEY>
 <JS>process.description;</JS></KEY></RE>
<RE key="state"/>: <RE><KEY><JS>process.STATE_PREFIX + process.state;
 </JS></KEY></RE>
<RE key="timeScheduled"/>: <RE key="readOnlyDateFormat"><PARM>
 <JS>if (process.scheduled != null) return process.scheduled.getTime();
 else return '';</JS></PARM></RE>
<RE key="timeCompleted"/>: <RE key="readOnlyDateFormat"><PARM>
 <JS>if (process.completed != null) return process.completed.getTime();
 else return '';</JS></PARM></RE>
<RE key="resultSummary"/>: <RE><KEY>
 <JS>process.STATE_PREFIX + process.resultSummary;
 </JS></KEY></RE>
<RE key="requester"/>: <JS>process.requestorName;</JS>
<RE key="requestedFor"/>: <JS>process.requesteeName;</JS>
<RE key="subject"/>: <JS>process.subject;</JS>
<RE key="comment"/>: <JS>process.comment;</JS>
<RE key="detail"/>: <JS>process.resultDetail;</JS>
```

Change Account Template

Provides information that the workflow activity has modified account information. By default, this template is disabled.

For example, the template provides this message:

```
Modified Account Information from IBM Security Identity Manager

The following ITIM Service [ITIM] account has been modified:

View Changes: http://localhost:9090/itim/console
Process Reference: 875016861865594505
Account ID: myaccount
Owner Name: firstname lastname
Time Completed: Jun 08, 2007 09:52:24 IST
```

The subject statement is:

```
<RE key="change_account_subject"/>
```

The plain text is:

```
<RE key="account_changed"><PARM>
 <RE key="service_name_with_profile_name"><PARM>
 <JS>EmailContext.getAccountServiceName();</JS></PARM>
 <PARM><RE><KEY><JS>EmailContext.getAccountServiceProfileName();
 </JS></KEY></RE></PARM></RE></PARM></RE>
<ITIMURL/>
<RE key="processRef"/>: <JS>process.id;</JS>
<JS>if (EmailContext.getTransactionId() != '0')
 { '<RE key="TRANSACTION_ID_LABEL"/>: ' + EmailContext.getTransactionId(); }
 </JS>
<RE key="accountID"/>: <JS>EmailContext.getAccountUserId();</JS>
<RE key="accountOwnerName"/>: <JS>EmailContext.getAccountOwnerName();</JS>
<RE key="timeCompleted"/>: <RE key="readOnlyDateFormat"><PARM>
 <JS>(new Date()).getTime();</JS></PARM></RE>
<JS>if (EmailContext.hasNewAccess()) { '<RE key="accountNewAccess"/>:
 <JS>EmailContext.getAccountNewAccessAsString();</JS>\n'; }</JS>
 <JS>if (EmailContext.hasRemovedAccess()) { '<RE key="accountRemovedAccess"/>:
 <JS>EmailContext.getAccountRemovedAccessAsString();</JS>\n'; }</JS>
 <JS>if (EmailContext.getTransactionId() != '0')
 { '<RE key="RETRIEVE_PASSWORD_TITLE"/>: ' +
  EmailContext.getPasswordRetrievalUrl(); }
 </JS>
```

```
<JS>if (EmailContext.getTransactionId() != '0')
 { '<RE key="passwordExpireLabel"/>:
<JS>if (EmailContext.getPasswordExpirePeriod() == 0)
 { '<RE key="passwordneverexpire"/>'; }
  else { EmailContext.getPasswordExpirePeriod(); }</JS>'; }</JS>
<JS>if (EmailContext.getTransactionId() != '0')
 { '<JS>if (EmailContext.getPasswordExpirePeriod() == 0)
 { '<RE key="additionalMsgForPwdRetrieval"/>'; }</JS>'; }</JS>
```

Compliance Template

Provides information that an account is not compliant with a provisioning policy. By default, this template is enabled.

For example, the template provides this message:

```
Compliance Alert for winlocal
Account [helpdesk35] is not compliant with the provisioning policy.
The value [Performance Log Users] of attribute [Local Groups]
should be [removed].
View Changes: http://99.99.999.99:80/itim/console
```

The subject statement is:

```
<RE key="compliance_alert_subject" >
<PARM><JS>var service = context.getService();
return service.getProperty("erservicename")[0];</JS>
</PARM>
</RE>
```

The plain text is:

```
<CAMessage/>
<RE key="itimUrl"/>:<ITIMURL/>
```

Delegation Template

Provides the default template for delegation, which includes the new delegation information. By default, this template is enabled and cannot be disabled. If any exception is thrown while evaluating JavaScript in the notification template or parsing the notification template, then the default delegation notification is sent.

For example, the template provides this message:

```
You have been selected to be the delegate:

For: John Doe

From: Tue Jul 03 08:00:13 IST 2012

To: Fri Jul 06 20:00:13 IST 2012

The subject statement is:
<RE key="delegationMailSubject"/>

The plain text is:
<RE key="delegationMailContent"/>

<RE key="delegationMailDelegator"/>:<JS>Delegate.getDelegator().name;</JS>

<RE key="delegationMailFrom"/>:<JS>Delegate.getStartDate();</JS>

<RE key="delegationMailTo"/>:<JS>Delegate.getEndDate();</JS>
```

Deprovision Account Template

Provides information that the workflow activity has removed an account. By default, this template is enabled.

For example, the template provides this message:

```
Your account has been removed by IBM Security Identity Manager.

The following Odessa Service [ADProfile] account has been deprovisioned.

View Changes: http://host:9080/itim/self
Process Reference: 5870349043636872731
```

```
Account ID: myaccount
Owner Name: myname
Reason: Policy Enforcement
Time completed: May 03, 2007 03:54:22 IST
```

The subject statement is:

```
<RE key="remove_account_subject" />
```

The plain text is:

```
 <RE key="account_deprovisioned">
  <PARM><RE key="service_name_with_profile_name">
  <PARM><JS>EmailContext.getAccountServiceName();</JS></PARM>
  <PARM><RE><KEY><JS>EmailContext.getAccountServiceProfileName();
  </JS></KEY></RE></PARM></RE></PARM></RE>
 <ITIMURL/>
 <RE key="processRef"/>: <JS>process.id;</JS>
 <RE key="accountID"/>: <JS>EmailContext.getAccountUserId();</JS>
 <RE key="accountOwnerName"/>: <JS>EmailContext.getAccountOwnerName();</JS>
 <RE key="reason"/>: <JS>EmailContext.getReason();</JS>
 <RE key="deprovisionCompleted"/>: <RE key="readOnlyDateFormat">
  <PARM><JS>(new Date()).getTime();</JS></PARM></RE>
```

Manual Activity Approval Template

Provides information that the user should provide information for a request. By default, this template is enabled.

For example, the template provides this message:

```
Pending workflow action: Case 884088984804067042.884090864796694775

You have been requested to submit information for the following request

View Changes: http://localhost:9090/itim/console
Description:
Requestee: firstname lastname
Subject: subject
Request Initiated: Jun 08, 2007 10:27:29 IST
Process Reference: 884088984804067042

Requested by process:
Process ID: 884066904196868932
Process Name: Provision Account
Description: Provisioning Account Process
Requester: System Administrator
Requestee: firstname lastname
Subject: subject
```

The subject statement is:

```
<RE key="pending_workitem_subject"><PARM><ID /></PARM></RE>
```

The plain text is:

```
 <RE key="wiApproval_message" />
 <ITIMURL/>
 <RE key="description"/>: <RE><KEY><JS>process.description;</JS></KEY></RE>
 <RE key="requestedFor"/>: <JS>process.requesteeName;</JS>
 <RE key="subject"/>: <JS>process.subject;</JS>
 <JS>if (process.subjectAccess!=null) if (process.subjectAccess.length>0)
  { '<RE key="accessName"/>: <JS>process.subjectAccess;</JS>\n'; }</JS>
  <RE key="requestInit"/>: <RE key="readOnlyDateFormat"><PARM>
  <JS>if (process.started != null) return process.started.getTime();
  else return '';</JS></PARM></RE>
 <RE key="processRef"/>: <JS>process.id;</JS>
 <JS>if (process.parentId == '0') { '<RE key="requestedBy"/>:
  <JS>process.requestorName;</JS>'; }</JS>

 <JS>if (process.parentId != '0') { '<RE key="parent_process"/>'; }</JS>
      <JS>if (process.parentId != '0')
    { '<RE key="processID"/>: ' + process.parentId; }</JS>
      <JS>if (process.parentId != '0') { '<RE key="processName"/>:
    <RE><KEY><JS>if (process.parentId != '0') { process.getParent().name; }
    </JS></KEY></RE>'; }</JS>
      <JS>if (process.parentId != '0') { '<RE key="description"/>:
    <RE><KEY><JS>if (process.parentId != '0')
```

```
    { process.getParent().description; } </JS></KEY></RE>'; }</JS>
      <JS>if (process.parentId != '0')
    { '<RE key="requester"/>: ' + process.getParent().requestorName; }
    </JS>
      <JS>if (process.parentId != '0')
    { '<RE key="requestedFor"/>: ' + process.getParent().requesteeName; }
   </JS>
      <JS>if (process.parentId != '0')
    { '<RE key="subject"/>: ' + process.getParent().subject; }</JS>
```

Manual Activity RFI Template

Provides the default template for request for information workflow activities. By default, this template is enabled

For example, the template provides this message:

```
You have been requested to submit information for the following request
http://localhost:9080/itim/self/ReviewActivities.do?
 activity=3053543743245419023
Description:
Requestee: Shoe Flower
Subject: shoe1
Request Initiated: Aug 03, 2007 11:48:52 IST
Process Reference: 3053543339468639238

Requested by process:
     Process ID: 3053541330639294422
     Process Name: Provision Account
     Description: Provision Account Process
     Requester: System Administrator
     Requestee: Shoe Flower
     Subject: shoe1
```

The subject statement is:

```
<RE key="pending_workitem_subject"><PARM><ID /></PARM></RE>
```

The plain text is:

```
 <RE key="wiRFI_message" />
 <ITIMURL/>
 <RE key="description"/>: <RE><KEY>
  <JS>process.description;</JS></KEY></RE>
 <RE key="requestedFor"/>: <JS>process.requesteeName;</JS>
 <RE key="subject"/>: <JS>process.subject;</JS>
 <JS>if (process.subjectAccess!=null)
  if (process.subjectAccess.length>0)
  { '<RE key="accessName"/>:
  <JS>process.subjectAccess;</JS>\n'; }</JS>
  <RE key="requestInit"/>: <RE key="readOnlyDateFormat"><PARM>
  <JS>if (process.started != null) return process.started.getTime();
  else return '';</JS></PARM></RE>
 <RE key="processRef"/>: <JS>process.id;</JS>
 <JS>if (process.parentId == '0') { '<RE key="requestedBy"/>:
  <JS>process.requestorName;</JS>'; }</JS>

 <JS>if (process.parentId != '0') { '<RE key="parent_process"/>'; }
  </JS>
      <JS>if (process.parentId != '0')
   { '<RE key="processID"/>: ' + process.parentId; }</JS>
      <JS>if (process.parentId != '0') { '<RE key="processName"/>:
   <RE><KEY><JS>if (process.parentId != '0') { process.getParent().name; }
   </JS></KEY></RE>'; }</JS>
      <JS>if (process.parentId != '0') { '<RE key="description"/>:
   <RE><KEY><JS>if (process.parentId != '0')
    { process.getParent().description; }
    </JS></KEY></RE>'; }</JS>
      <JS>if (process.parentId != '0')
    { '<RE key="requester"/>: ' + process.getParent().requestorName; }
    </JS>
      <JS>if (process.parentId != '0')
    { '<RE key="requestedFor"/>: ' + process.getParent().requesteeName; }
    </JS>
      <JS>if (process.parentId != '0')
    { '<RE key="subject"/>: ' + process.getParent().subject; }</JS>
```

Manual Activity Work Order Template

Provides default template for the work order workflow manual activity. By default, this template is enabled.

For example, the template provides this message:

```
Pending workflow action:
 Case 14019933646588032 75.1402011582339065124

You have received a Work Order request
```

The subject statement is:

```
<RE key="pending_workitem_subject"><PARM><ID /></PARM></RE>
```

The plain text is:

```
 <RE key="wiWorkOrder_message" />
```

New Account Template

Provides information that the workflow activity has created a new account. By default, this template is enabled.

For example, the template provides this message:

```
New Account Information from IBM Security Identity Manager

The following new ITIM Service [ITIM] account has been created for you:

View Changes: http://localhost:80/itim/console
Process Reference: 8498649245880216244
Password: bAMI#gai
Account ID: myaccount
Owner Name: firstname lastname
Time of service provision: Jun 29, 2007 10:55:58 IST
```

The subject statement is:

```
<RE key="new_account_subject"/>
```

The plain text is:

```
 <RE key="account_created"><PARM>
  <RE key="service_name_with_profile_name">
  <PARM><JS>EmailContext.getAccountServiceName();</JS></PARM>
  <PARM><RE><KEY><JS>EmailContext.getAccountServiceProfileName();
  </JS></KEY></RE></PARM></RE></PARM></RE>
 <ITIMURL/>
 <RE key="processRef"/>: <JS>process.id;</JS>
 <JS>if (EmailContext.getTransactionId() != '0')
  { '<RE key="TRANSACTION_ID_LABEL"/>: '
  + EmailContext.getTransactionId(); } </JS>
 <RE key="password"/>: <JS>EmailContext.getAccountPassword();</JS>
 <RE key="accountID"/>: <JS>EmailContext.getAccountUserId();</JS>
 <RE key="accountOwnerName"/>:
  <JS>EmailContext.getAccountOwnerName();</JS>
 <RE key="timeofprovision"/>: <RE key="readOnlyDateFormat">
  <PARM><JS>(new Date()).getTime();</JS></PARM></RE>
 <JS>if (EmailContext.hasNewAccess()) { '<RE key="accountNewAccess"/>:
  <JS>EmailContext.getAccountNewAccessAsString();</JS>\n'; }</JS>
 <JS>if (EmailContext.getTransactionId() != '0')
  { '<RE key="RETRIEVE_PASSWORD_TITLE"/>: '
   + EmailContext.getPasswordRetrievalUrl(); }</JS>
 <JS>if (EmailContext.getTransactionId() != '0')
  { '<RE key="passwordExpireLabel"/>:
  <JS>if (EmailContext.getPasswordExpirePeriod() == 0)
  { '<RE key="passwordneverexpire"/>'; }
  else { EmailContext.getPasswordExpirePeriod(); }</JS>'; }</JS>
 <JS>if (EmailContext.getTransactionId() != '0')
  { '<JS>if (EmailContext.getPasswordExpirePeriod() == 0)
  { '<RE key="additionalMsgForPwdRetrieval"/>'; }</JS>'; }</JS>
```

New Password Template

Provides information that there is a new password for an account. By default, this template is enabled.

For example, the template provides this message:

```
Account new password information

The following is your new password for account myaccount:

View Changes: http://localhost:9090/itim/console
Process Reference: 2855285841498421007
New Password: secret
Account ID: myaccount
Account Service: ITIM Service
Account Service Profile: ITIM
Owner Name: firstname lastname
Time of service provision: Apr 25, 2007 12:54:05 IST
```

The subject statement is:

```
<RE key="password_change_subject"/>
```

The plain text is:

```
 <RE><KEY><JS>if (EmailContext.getTransactionId() == '0')
  { 'newAccountPassword' } else { 'newAccountPasswordPickUp'; }
   </JS></KEY>
  <PARM><JS>process.subject;</JS></PARM></RE>
 <ITIMURL/>
 <RE key="processRef"/>: <JS>process.id;</JS>
 <JS>if (EmailContext.getTransactionId() != '0')
  { '<RE key="TRANSACTION_ID_LABEL"/>: ' +
   EmailContext.getTransactionId(); }
  </JS>
 <RE key="newPassword"/>: <JS>EmailContext.getAccountPassword();</JS>
 <RE key="accountID"/>: <JS>EmailContext.getAccountUserId();</JS>
 <RE key="accountService"/>:
  <JS>EmailContext.getAccountServiceName();</JS>
 <RE key="accountServiceProfile"/>: <RE><KEY>
  <JS>EmailContext.getAccountServiceProfileName();</JS></KEY></RE>
 <RE key="accountOwnerName"/>:
  <JS>EmailContext.getAccountOwnerName();</JS>
 <RE key="timeofprovision"/>: <RE key="readOnlyDateFormat">
  <PARM><JS>(new Date()).getTime();</JS></PARM></RE>
 <JS>if (EmailContext.getTransactionId() != '0')
  { '<RE key="RETRIEVE_PASSWORD_TITLE"/>: '
  + EmailContext.getPasswordRetrievalUrl(); }</JS>
 <JS>if (EmailContext.getTransactionId() != '0')
  { '<RE key="passwordExpireLabel"/>:
  <JS>if (EmailContext.getPasswordExpirePeriod() == 0)
  { '<RE key="passwordneverexpire"/>'; }
  else { EmailContext.getPasswordExpirePeriod(); }</JS>'; }</JS>

 <JS>if (EmailContext.getTransactionId() != '0')
  { '<JS>if (EmailContext.getPasswordExpirePeriod() == 0)
  { '<RE key="additionalMsgForPwdRetrieval"/>'; }</JS>'; }</JS>
```

Process Completion Template

Provides information that the workflow activity has completed. By default, this template is enabled.

For example, the template provides this message when an activity is completed without being canceled:

```
A workflow process, 1416721862784240178, has completed.
Result Summary: Success
The following process has completed

Process Information

View Changes: http://localhost:9090/itim/console
Process ID: 1416721862784240178
Activity:
Description: Modify Provisioning Policy Process
State: Completed
Date submitted: May 16, 2007 12:22:58 IST
Time Completed: May 16, 2007 01:44:17 IST
Result Summary: Success
Requester: System Administrator
```

```
Requestee:
Subject: Default Provisioning Policy for service Win Local Profile
Comment:
Detail:
```

For example, the template provides this message when an activity is canceled:

```
Subject: A workflow process, 6690130336188564930, has completed.
Result Summary: Failed
The following process has completed

Process Information

View Changes: http://localhost:80/itim/console
Process ID: 6690130336188564930
Activity: Person Add
Description: Person Add Process
State: Canceled
Date submitted: Jan 30, 2014 01:13:59 CST
Time Completed: Jan 29, 2014 01:13:22 CST
Result Summary: Failed
Requester: System Administrator
Requestee: firstname lastname
Subject:
Comment:
Detail:
Canceled By: System Administrator
Date Canceled:  Jan 29, 2014 01:13:22 CST
Canceled Justification: No longer needed
```

The subject statement is:

```
<RE key="processCompletedSubject"><PARM><JS>process.id;</JS></PARM>
 <PARM><RE key="resultSummaryValue"><PARM><RE><KEY>
 <JS>process.STATE_PREFIX + process.resultSummary;
 </JS></KEY></RE></PARM></RE></PARM></RE>
```

The plain text is:

```
 <RE key="process_completed_message" />

 <RE key="processInformation" />
 <ITIMURL/>
 <RE key="processID"/>: <JS>process.id;</JS>
 <RE key="name"/>: <RE><KEY><JS>process.name;</JS></KEY></RE>
 <RE key="description"/>: <RE><KEY><JS>process.description;</JS>
  </KEY></RE>
 <RE key="state"/>: <RE><KEY>
  <JS>process.STATE_PREFIX + process.state;</JS></KEY></RE>
 <RE key="timeScheduled"/>: <RE key="readOnlyDateFormat"><PARM>
  <JS>if (process.scheduled != null)
  return process.scheduled.getTime();
  else return '';</JS></PARM></RE>
 <RE key="timeCompleted"/>: <RE key="readOnlyDateFormat"><PARM>
  <JS>if (process.completed != null)
  return process.completed.getTime();
  else return '';</JS></PARM></RE>
 <RE key="resultSummary"/>: <RE><KEY>
  <JS>process.STATE_PREFIX + process.resultSummary;</JS>
  </KEY></RE>
 <RE key="requester"/>: <JS>process.requestorName;</JS>
 <RE key="requestedFor"/>: <JS>process.requesteeName;</JS>
 <RE key="subject"/>: <JS>process.subject;</JS>
 <RE key="comment"/>: <JS>process.comment;</JS>

 <RE key="detail"/>: <JS>process.resultDetail;</JS>
<JS>if (process.cancelor_name != null)
  { '<RE key="CanceledBy"/>: ' + process.cancelor_name; }</JS>
<JS>if (process.cancelor_name != null)
  { '<RE key="DateCanceled"/>: '; }</JS>
<RE key="readOnlyDateFormat"><PARM>
<JS>if (process.canceled_date != null) return process.canceled_date.getTime();
  else return '';</JS>
</PARM></RE>
```

```
<JS>if (process.cancelor_name != null) { '<RE key="CanceledReason"/>:
<JS>if (process.canceled_justification == null) { return ' '; }
  else { return process.canceled_justification;}
</JS>'; }</JS>
```

Process Timeout Template

Provides information that the workflow process has timed out. By default, this template is enabled.

For example, the template provides this message:

```
Workflow activity is being timed out and will be
 terminated by the workflow system

Activity Information
View Changes: http://localhost:9080/itim/console
Activity ID: RECERTAPPROVAL
Activity: $ITIM_RECERTIFY
Time Started: Aug 02, 2007 03:18:54 IST
Time Completed:
Result Summary: Pending
State: Running
Activity Type: Manual Approval/Reject

Process Information

Process ID: 8566433417513336819
Activity: Recertification of Account/Access
Description: Recertification of Account/Access
State: Running
Date submitted: Aug 02, 2007 03:18:54 IST
Time Completed:
Result Summary:
Requester: org
Requestee: Person B
Subject: personb
Comment:
Detail:
```

The subject statement is:

```
<RE key="process_timeout_subject" />
```

The plain text is:

```
 <RE key="process_timeout_message" />

 <RE key="processInformation" />
 <ITIMURL/>
 <RE key="processID"/>: <JS>process.id;</JS>
 <RE key="name"/>: <RE><KEY><JS>process.name;</JS></KEY></RE>
 <RE key="description"/>: <RE><KEY><JS>process.description;</JS></KEY></RE>
 <RE key="state"/>: <RE><KEY>
  <JS>process.STATE_PREFIX + process.TIMEOUT;</JS></KEY></RE>
 <RE key="timeScheduled"/>: <RE key="readOnlyDateFormat"><PARM>
  <JS>if (process.scheduled != null) return process.scheduled.getTime();
  else return '';</JS></PARM></RE>
 <RE key="timeCompleted"/>: <RE key="readOnlyDateFormat"><PARM>
  <JS>if (process.completed != null) return process.completed.getTime();
  else return '';</JS></PARM></RE>
 <RE key="resultSummary"/>: <RE><KEY>
  <JS>process.STATE_PREFIX + process.resultSummary;</JS></KEY></RE>
 <RE key="requester"/>: <JS>process.requestorName;</JS>
 <RE key="requestedFor"/>: <JS>process.requesteeName;</JS>
 <RE key="subject"/>: <JS>process.subject;</JS>
 <RE key="comment"/>: <JS>process.comment;</JS>

 <RE key="detail"/>: <JS>process.resultDetail;</JS>
```

Restore Account Template

Provides information that an account has been restored. By default, this template is enabled.

For example, the template provides this message:

```
Restored Account Information from IBM Security Identity Manager

The following ITIM Service [ITIM] account has been restored:

View Changes: http://localhost:9090/itim/console
Process Reference: 2857890686820910405
New Password: secret
Account ID: myaccount
Owner Name: firstname lastname
Time Completed: Apr 25, 2007 01:04:08 IST
```

The subject statement is:

```
<RE key="restore_account_subject"/>
```

The plain text is:

```
  <RE key="restore_account"><PARM>
  <RE key="service_name_with_profile_name"><PARM>
  <JS>EmailContext.getAccountServiceName();</JS></PARM>
  <PARM><RE><KEY>
  <JS>EmailContext.getAccountServiceProfileName();
  </JS></KEY></RE></PARM></RE></PARM></RE>
 <ITIMURL/>
 <RE key="processRef"/>: <JS>process.id;</JS>
 <JS>if (EmailContext.getTransactionId() != '0')
  { '<RE key="TRANSACTION_ID_LABEL"/>: '
  + EmailContext.getTransactionId(); } </JS>
 <RE key="newPassword"/>: <JS>EmailContext.getAccountPassword();</JS>
 <RE key="accountID"/>: <JS>EmailContext.getAccountUserId();</JS>
 <RE key="accountOwnerName"/>:
  <JS>EmailContext.getAccountOwnerName();</JS>
 <RE key="timeCompleted"/>: <RE key="readOnlyDateFormat">
 <PARM>
  <JS>(new Date()).getTime();</JS></PARM></RE>
 <JS>if (EmailContext.getTransactionId() != '0')
  { '<RE key="RETRIEVE_PASSWORD_TITLE"/>: '
  + EmailContext.getPasswordRetrievalUrl(); }</JS>
 <JS>if (EmailContext.getTransactionId() != '0')
  { '<RE key="passwordExpireLabel"/>:
 <JS>if (EmailContext.getPasswordExpirePeriod() == 0)
  { '<RE key="passwordneverexpire"/>'; }
  else { EmailContext.getPasswordExpirePeriod(); }</JS>'; }
  </JS>

 <JS>if (EmailContext.getTransactionId() != '0')
  { '<JS>if (EmailContext.getPasswordExpirePeriod() == 0)
  { '<RE key="additionalMsgForPwdRetrieval"/>'; }</JS>'; }</JS>
```

Suspend Account Template

Provides information that an account is suspended. By default, this template is enabled.

For example, the template provides this message:

```
Your account has been suspended by IBM Security Identity Manager

The following AD Service (RFI) [ADProfile] account has been suspended:

View Changes: http://localhost:9090/itim/console
Process Reference: 2857497715286893521
Account ID: myaccount
Owner Name: firstname lastname
Time Completed: Apr 25, 2007 01:02:43 IST
```

The subject statement is:

```
<RE key="suspend_account_subject" />
```

The plain text is:

```
 <RE key="account_suspended"><PARM>
  <RE key="service_name_with_profile_name">
  <PARM><JS>EmailContext.getAccountServiceName();</JS></PARM>
  <PARM><RE><KEY><JS>EmailContext.getAccountServiceProfileName();
  </JS></KEY></RE></PARM></RE></PARM></RE>
 <ITIMURL/>
```

```
<RE key="processRef"/>: <JS>process.id;</JS>
<RE key="accountID"/>: <JS>EmailContext.getAccountUserId();</JS>
<RE key="accountOwnerName"/>:
 <JS>EmailContext.getAccountOwnerName();</JS>
<RE key="timeCompleted"/>: <RE key="readOnlyDateFormat"><PARM>
 <JS>(new Date()).getTime();</JS></PARM></RE>
```

To-Do Reminder Template

Provides the default template for workflow reminders, which are email messages that remind users about pending activities to which they not responded. By default, this template is disabled.

For example, the template provides this message:

```
Subject: Pending workflow action:
 Case 6167063972298972180.6167064647650050990

The following request has been submitted for your approval
View Changes: http://localhost:9080/itim/console
Description: ApprovalWorkflow
Requestee: firstname lastname
Subject: subject
Request Initiated: Sep 05, 2007 05:42:18 IST
Process Reference: 6167063972298972180


Requested by process:
     Process ID: 6167052766519381908
     Process Name: Provision Account
     Description: Provision Account Process
     Requester: System Administrator
     Requestee: firstname lastname
     Subject: subject

 This WorkItem will be escalated on: Saturday, September 8, 2007.
```

The subject statement is:

```
<originalSubject/>
```

The plain text is:

```
 <textBody/>

  <RE key="escalation_note"/> <escalationTime/>
```

Chapter 9. JavaScript extensions overview

JavaScript is used in IBM Security Identity Manager to specify identity policies, provisioning policy parameters, service selection policies, placement rules for identity feeds, and orphan account adoption.

In addition, JavaScript is used in workflows to specify transition conditions, loop conditions, JavaScript activities, activity postscripts, and workflow notification. Various scripting extensions are provided by IBM Security Identity Manager to expose useful data and services to each of these scripts. In addition to these extensions, system administrators can configure IBM Security Identity Manager to load custom JavaScript extensions. For more information about custom JavaScript extensions, see the `scriptframework.properties` file. Use the Update Property page from the **Appliance Dashboard** of the IBM Security Identity Manager virtual appliance console. See Managing the server properties.

IBM Security Identity Manager supports the IBM `JSEngine` Java Script interpreter. This interpreter supports the third edition (December 1999) of the ECMA-262 specification.

Table 7 lists the supported host components and script extensions.

Table 7. Host components and script extensions

Host Component	Supported Script Extension	Description
`AccountTemplate`	`ProvisioningPolicyExtension`	Extensions registered with this key are loaded by Account Default Template parameters.
	`ServiceExtension`	
	`SubjectExtension`	
`Delegate`	`DelegateExtension`	Extensions registered with this key are loaded by Delegation notifications.
	`Model Extensions Package`	
`HostSelection`	`ServiceModelExtension`	Extensions registered with this key are loaded by Service Selection Policies.
	`SubjectExtension`	
`IdentityPolicy`	`IdentityPolicyExtension`	Extensions registered with this key are loaded by Identity Policies.
	`AttributesExtension`	
	`ServiceExtension`	
	`SubjectExtension`	
`OrphanAdoption`	Model Extensions Package	Extensions registered with this key are loaded by adoption scripts.
	`ServiceExtension`	
	`SubjectExtension`	
`PersonPlacementRules`	`PersonPlacementRulesExtension`	Extensions registered with this key are loaded by placement rules during identity feeds.
	`ServiceExtension`	
	`AttributesExtension`	
`PostOffice`	`PostOfficeExtension`	Extensions registered with this key are loaded by Post Office templates.

Table 7. Host components and script extensions (continued)

Host Component	Supported Script Extension	Description
ProvisioningPolicy	ProvisioningPolicyExtension	Extensions registered with this key are loaded by Provisioning Policy parameters.
	Model Extensions Package	
	ServiceExtension	
	SubjectExtension	
	AttributesExtension (deprecated)	
Reminder	ReminderExtension	Extensions registered with this key are loaded by email reminder templates.
	SubjectExtension	
Workflow	WorkflowExtension	Extensions registered with this key are loaded by workflow scripts that include Workflow TODO Notifications.
	Model Extensions Package	
	LoopCountExtension	
Workflow Notification	WorkflowExtension	The extensions loaded are hardcoded and all supported extensions are loaded.
	EmailContextExtension	
	PersonModelExtension	
TODO Notification (Approval/RFI/ComplianceAlert/ WorkOrder)	WorkflowExtension	The extensions loaded are the same as Workflow.
	Model Extensions Package	
	LoopCountExtension	

Packaged extensions

The section describes the scripting extensions provided by IBM Security Identity Manager, the JavaScript objects they expose, and the scripts to which these extensions are applicable.

Do not remove these extensions from the properties file that you configure, because they are necessary for standard product operation. All of the extensions are configured for new installations.

AttributesExtension

The full extension name is com.ibm.itim.script.extensions.AttributesExtension.

This extension is responsible for making the ATTRIBUTES object available to scripts. ATTRIBUTES is a Map type object and is used internally to implement the deprecated Enrole.getAttributeValue() and Enrole.getAttributeValues() methods.

AttributesExtension and the ATTRIBUTES script object are deprecated. Do not use them in any new scripts.

Availability

IdentityPolicy
PersonPlacementRules
ProvisioningPolicy

JavaScript Objects

ATTRIBUTES

DelegateExtension

The full extension name is `com.ibm.itim.script.extensions.DelegateExtension`.

This extension is responsible for making the `Delegate` object available to delegation notification scripts.

Availability

`Delegation Notification`

JavaScript Objects

`Delegate`

EmailContextExtension

The full extension name is `com.ibm.itim.workflow.script.EmailContextExtension`.

The `EmailContextExtension` provides the `EmailContext` object that provides information about the workflow activity and process that is making the notification. `EmailContext` is of type `WorkflowRuntimeContext`, although it might be a more specific subtype, depending on what type of change triggered the notification.

Availability

Notification

JavaScript Objects

`EmailContext`

EnroleExtension

The full extension name is `com.ibm.itim.script.extensions.EnroleExtension`.

This extension is automatically loaded for all scripts. It is not in the scriptframework.properties file.

This extension exposes the `Enrole` object to scripts. This object provides the following miscellaneous functions:

- Converting to and from the generalized time format.
- Logging and tracing facilities to write to the Security Identity Manager logs.

Availability

All scripts

JavaScript Objects

`Enrole`
`Error`

IdentityPolicyExtension

The full extension name is `com.ibm.itim.policy.script.IdentityPolicyExtension`.

This extension exposes the `IdentityPolicy` object to identity policy scripts. This object provides a method to test for the existence of a user ID.

Availability

Identity Policy

JavaScript Objects

IdentityPolicy

LoopCountExtension

The full extension name is com.ibm.itim.workflow.script.LoopCountExtension.

This extension provides the loopcount script object. The object is an integer that tells a script how many times a loop ran.

Availability

Workflow

JavaScript Objects

loopcount

Model extensions package

The model extensions expose JavaScript objects that can be used to search for people, accounts, services, and organizational units such as organizations, business units, and locations.

Important: The objects exposed by these extensions allow access to identity and service data without regard to specified access control rules for these data. The objects are considered privileged. Define access control items that manage access to IBM Security Identity Manager scripts.

All of the model extensions have the same availability and can be used with the following extension points:

- AccountTemplate
- ProvisioningPolicy
- HostSelection
- OrphanAdoption
- Workflow
- Notification

AccountModelExtension

The full extension name is com.ibm.itim.script.extensions.model.AccountModelExtension.

This extension exposes the Account constructor and AccountSearch constructor to applicable scripts. After it is constructed, an Account object represents an Account Directory Object in scripts. The AccountSearch object provides methods to search for existing accounts based on several parameters, which include **uid**, **owner**, and **service**.

JavaScript Objects

- AccountSearch
- Account

PersonModelExtension

The full extension name is com.ibm.itim.script.extensions.model.PersonModelExtension.

This extension exposes the Person constructor, PersonSearch constructor, and ExtendedPerson constructor to applicable scripts. After it is constructed, a Person object represents a Person Directory Object in script. A ExtendedPerson object

extends Person with ownership type information. The PersonSearch object provides methods to search for existing people based on a provided LDAP filter.

JavaScript Objects

- PersonSearch
- Person
- ExtendedPerson

OrganizationModelExtension

The full extension name is com.ibm.itim.script.extensions.model.OrganizationModelExtension.

This extension exposes the ContainerSearch constructor to applicable scripts. The ContainerSearch object provides methods to search of Organizational containers based on LDAP filters.

JavaScript Objects

ContainerSearch

RoleModelExtension

The full extension name is com.ibm.itim.script.extensions.model.RoleModelExtension.

This extension exposes the Role constructor and RoleSearch constructor to applicable scripts. After it is constructed, the Role object represents a Role Directory Object in scripts. The RoleSearch object provides a method to search for Roles based on role name.

JavaScript Objects

- RoleSearch
- Role

ServiceModelExtension

The full extension name is com.ibm.itim.script.extensions.model.ServiceModelExtension.

This extension exposes the Service constructor and ServiceSearch constructor to applicable scripts. After it is constructed the Service object represents a Service Directory Object in scripts. The ServiceSearch object provides methods to search for Service based on several parameters, which include LDAP filter and service name.

JavaScript Objects

- ServiceSearch
- Service

PersonPlacementRulesExtension

The full extension name is com.ibm.itim.remoteservices.script.PersonPlacementRulesExtension.

This extension provides the entry object to the scripting environment. The entry object is of type Map and contains the attribute values for the Person that is placed.

Availability

PersonPlacementRules

JavaScript Objects

`entry`

PostOfficeExtension

The full extension name is `com.ibm.itim.mail.postoffice.script.PostOfficeExtension`.

The Post Office capability reduces the number of email messages received by workflow participants by combining similar notifications into a single email. The emails are combined with a template specified in the system configuration pages. This extension exposes a JavaScript object, `PostOffice`, to JavaScript snippets in these templates. This object provides methods for accessing all the distinct emails, the email address of the recipient, the email topic, and the recipient data.

Availability

Post Office Template

JavaScript Objects

`PostOffice`

ProvisioningPolicyExtension

The full extension name is`com.ibm.itim.policy.script.ProvisioningPolicyExtension`.

This extension provides the scripting objects reason and parameters to the scripting environment. The reason object is an integer that informs a script of the reason the evaluation is happening: 0 if a new account or 1 if an existing account. The parameters object is a map that contains the information about the account that is being evaluated. Currently, only the **`uid`** field is supported.

Availability

`AccountTemplate`
`ProvisioningPolicy`

JavaScript Objects

`parameters`
`reason`

ReminderExtension

The full extension name is `com.ibm.itim.script.extensions.ReminderExtension`.

This extension exposes the `reminderCtx` object to JavaScript snippets contained in email reminders. This object provides methods for accessing the original email text and subject. It also provides the due date and time for the associated to-do item.

Availability

`E-mail reminders`

JavaScript Objects

`reminderCtx`

ServiceExtension

The full extension name is `com.ibm.itim.script.extensions.ServiceExtension`.

This extension exports the service object to the scripting environment. The service object is a `DirectoryObject` type and represents the Service associated with a provisioning operation.

Availability

`IdentityPolicy`
`OrphanAdoption`
`PersonPlacementRules`
`AccountTemplate`
`ProvisioningPolicy`

JavaScript Objects

`service`

SubjectExtension

The full extension name is `com.ibm.itim.script.extensions.SubjectExtension`.

This extension provides the subject scripting object. In all of the scripting contexts except for OrphanAdoption, subject is a DirectoryObject. In the OprhanAdoption context, subject is a Map of the attributes returned by a reconciliation.

Availability

HostSelection
IdentityPolicy
OrphanAdoption
Reminder
AccountTemplate
ProvisioningPolicy

JavaScript Objects

subject

WorkflowExtension

The full extension name is `com.ibm.itim.workflow.script.WorkflowExtension`.

This extension exposes JavaScript objects that can be used to access data from a workflow process in progress. In addition, it exposes objects that can be used to get or set the status, state, and result of a workflow process or activity.

Availability

`Workflows`

JavaScript Objects

- `process`
- `activity`
- `participant`
- `Relevant Data`

  Note: `Relevant Data` are objects defined by the workflow designer. Check with system administrator to find the names of specific `Relevant Data` objects.

Relevant data JavaScript objects

Each process has a set of relevant data, or process specific parameters, which can be read or changed from in a workflow script.

The name and syntax of these parameters, or relevant data items, are defined in the workflow designer and are typically specific to the workflow process purpose. For example when you add a user, an object that holds all the attributes of the new user can be a relevant data item. However, when you delete a user, the only required relevant data item can be the distinguished name of the user to delete.

Each relevant data item is represented in the workflow script as a variable with the same relevant data ID as defined in the workflow designer. These relevant data items all have the following functions:

get() This function returns a JavaScript object that represents the value of the relevant data item. There is a variable present for each relevant data item in the context of the script. For performance reasons, however, the values are not retrieved from the workflow engine until the script specifically requests it with this call. The returned JavaScript object is in the same syntax as defined in the workflow designer.

Usage:

```
dn = subjectDN.get();
```

where `subjectDN` is defined as a relevant data item for the current process.

set(Object value)
The `set(Object value)` function changes the value of the relevant data item. It not only updates the relevant data item in the script, but also in the workflow engine. The new value is a parameter to the function. The new value must be compatible with the syntax of the relevant data item as defined in the workflow designer. For example, if the relevant data item is an integer, the value `cat` is not a valid parameter to this function.

Usage:

```
ou.set("engineering");
```

where `ou` is defined as a relevant data item for the current process.

Registering JavaScript extensions

JavaScript extensions might not be useful or applicable to every scriptable function that IBM Security Identity Manager provides. For example, an extension used by Post Office templates might not be applicable to provisioning policy parameters. An extension designed for one class of script might not load or behave appropriately when loaded into another class of script.

Security Identity Manager has the classes of script that are listed in Table 8 on page 67. JavaScript extensions might be registered to load and run with any combination of these script classes.

JavaScript extensions are configured in these files:

scriptframework.properties (suggested)
For *all* new extensions. Use this file to configure script extensions and other scripting functions.

JavaScript extensions are registered in the `scriptframework.properties` file. This file is formatted with the standard Java Properties *key*[.*subkey*]=*value* format.

- The `key` is the name assigned to the target script class, described in Table 8 on page 67.

- The `value` is the full class name of the `ScriptExtension` interface.
- (Optional) The `subkey` is used when more than one extension is registered for a script class.

Use the Update Property page from the **Appliance Dashboard** of the IBM Security Identity Manager virtual appliance console. See Managing the server properties.

Note:

1. To prevent the possibility of a code injection attack, do not use the JavaScript function `eval()`.
2. By default, only the set of extensions registered in the `scriptframework.properties` file is available for the particular script. You can configure any supported extension for the script by registering JavaScript extensions in the `scriptframework.properties` file. For information about supported script extensions, see Table 7 on page 59. For information about the properties and methods available for each JavaScript extension object, see Chapter 10, "JavaScript extension reference," on page 75.

The following line registers a single extension for use in Security Identity Manager scripts:

```
ITIM.extension.IdentityPolicy=com.ibm.itim.policy.script.IdentityPolicyExtension
```

These example lines register multiple extensions for use in Security Identity Manager scripts:

```
ITIM.extension.IdentityPolicy.1=com.ibm.itim.policy.script.IdentityPolicyExtension
ITIM.extension.IdentityPolicy.2=com.yourcompany.script.YourCustomExtension
```

Table 8. Script class keys

Host Component	Script Class Key
AccountTemplate	ITIM.extension.AccountTemplate
Delegate	ITIM.extension.Delegate
HostSelection	ITIM.extension.HostSelection
IdentityPolicy	ITIM.extension.IdentityPolicy
OrphanAdoption	ITIM.extension.OrphanAdoption
PersonPlacementRules	ITIM.extension.PersonPlacementRules
PostOffice	ITIM.extension.PostOffice
ProvisioningPolicy	ITIM.extension.ProvisioningPolicy
Reminder	ITIM.extension.Reminder
Workflow	ITIM.extension.Workflow
Workflow Notification	ITIM.extension.Notification
TODO Notification (Approval/RFI/ ComplianceAlert/WorkOrder)	ITIM.extension.Notification

Configuring scriptframework.properties

Use the `scriptframework.properties` file, which provides extended documentation for these tasks, to configure major scripting functions.To update the file, use the Update Property page from the **Appliance Dashboard** of the IBM Security Identity Manager virtual appliance console. See Managing the server properties.

Following are the major scripting functions:

Extensions
Specifies which extensions to load for each host component. To load more than a single extension for any host component, add a suffix to the properties key (each key must be unique). For example:

```
ITIM.extension.IdentityPolicy=com.ibm.itim.policy.script.IdentityPolicyExtension
ITIM.extension.IdentityPolicy.service=com.ibm.itim.script.extensions.ServiceExtension
```

Interpreters
Configures the interpreter to use for each host component. The default is the IBM JSEngine.

Wrappers
All objects available to scripts are really Java objects that are used by IBM Security Identity Manager. To prevent security issues, IBM Security Identity Manager wraps these objects in wrappers. Use this area of the `scriptframework.properties` file to change the default wrappers that are used by IBM Security Identity Manager. Default scripts that are provided by IBM Security Identity Manager assume the use of default wrappers. If you change the scripts, functions might stop working. This area is for advanced use only.

Miscellaneous
Determines whether profiling information is collected and included in the `trace` log and whether plain text passwords can be accessed from `Person` and `Account` objects.

Migration of custom FESI extensions to the IBM JSEngine

Migration of a custom FESI extension to a script extension makes your code shorter, easier to read, and easier to understand.

Note: Support for FESI is deprecated in IBM Security Identity Manager Version 6.0.

For detailed information and examples about how to write new extensions, obtain a copy of the `extensions.zip` file. Do these steps:

1. Log on to the IBM Security Identity Manager virtual appliance console to open the **Appliance Dashboard**.
2. From the top-level menu of the Appliance Dashboard, select **Configure** > **Advanced Configuration** > **Custom File Management** to display the Custom File Management page.
3. Click the **All Files** tab.
4. Go to `directories/utilities`.
5. Select `extensions.zip` and click **Download**.
6. Extract the `extensions.zip` file.
7. See the documentation in `/extensions/doc/javascript/javascript.html`.

The following example illustrates the migration steps.

Best practice in handling function returns

You can minimize problems that might occur due to differences in how FESI and IBM JSEngine handle JavaScript. The differences involve implicit return values from functions.

For example, given these statements:

```
function sumValue() {
  var a = 3;
  var b = 2;
  a + b;
}
```

With FESI, the function `sumValue()` returns 5 because 5 is the result of the last statement run in the function. Using IBM JSEngine, the expression `sumValue()` returns null because there is no explicit return. The correct code for IBM JSEngine includes an explicit return statement:

```
function sumValue() {
  var a = 3;
  var b = 2;
  return a + b;
}
```

To keep JavaScript code consistent, always use an explicit return value in functions. In the previous release, some of the service selection script examples did not use an explicit return value. Update any JavaScript code that is based on these examples to have an explicit return value, to ensure that the code continues to work after an upgrade to use IBM JSEngine.

Plain Old Java Object (POJO) example

Start with a Plain Old Java Object (POJO, in this example) that contains all of the business logic for your extension.

For example:

```
public class Extension {
   public static void log(String msg) {
      System.out.println(msg);
   }
}
```

In this case, the POJO contains a single method. Your typical extension contains more logic. For example:

```
static class FESIExtension implements JSExtension {
   public void initializeExtension(JSGlobalObject go) throws JSException {
      // Create the prototype
      final JSObject prototype = go.makeJSObject();

      prototype.setMember("log", new JSFunctionAdapter() {
         public Object doCall(JSObject thisObject, Object[] args)
               throws JSException {
            if (args.length >= 1) {
               Extension.log(args[0].toString());
            }

            return null;
         }
      });

      final JSObject obj = go.makeJSObject(prototype);

      // This is the name of the object to be used in JavaScript Code
      go.setMember("CustomExtension", obj);

      go.setMember("log", new JSFunctionAdapter() {
         public Object doCall(JSObject thisObject, Object[] args)
               throws JSException {
            if (args.length >= 1) {
```

```
            Extension.log(args[0].toString());
         }

         return null;
      }
   });

   go.setMember("Logger", new JSFunctionAdapter() {
      public Object doNew(JSObject thisObject, Object[] args)
            throws JSException {
         JSGlobalObject go = thisObject.getGlobalObject();
         JSObject proto = go.makeJSObject();

         proto.setMember("log", new JSFunctionAdapter() {
            public Object doCall(JSObject thisObject, Object[] args)
                  throws JSException {
               if (args.length >= 1) {
                  Extension.log(args[0].toString());
               }

               return null;
            }
         });
         final JSObject obj = go.makeJSObject(proto);
         return obj;
      }
   });
  }
}
```

This FESI extension has three main parts:

1. First, the extension makes a `JSObject` named prototype and adds the method "log" to prototype:

   ```
   final JSObject prototype = go.makeJSObject();

   prototype.setMember("log", new JSFunctionAdapter() {
      public Object doCall(JSObject thisObject, Object[] args)
            throws JSException {
         if (args.length >= 1) {
            Extension.log(args[0].toString());
         }

         return null;
      }
   });

   go.setMember("CustomExtension", obj);
   ```

   The prototype `JSObject` is then added to the `JSGlobalObject` with the name `CustomExtension`. This addition allows scripts to call:

   ```
   CustomExtension.log("message");
   ```

2. The second part of the extension creates a global function named `log`.

   ```
   go.setMember("log", new JSFunctionAdapter() {
      public Object doCall(JSObject thisObject, Object[] args)
            throws JSException {
         if (args.length >= 1) {
            Extension.log(args[0].toString());
         }

         return null;
      }
   });
   ```

   Now, a script can call:

   ```
   log("message");
   ```

3. The third part of the extension creates a constructor that can be called from scripts. For example:

```
go.setMember("Logger", new JSFunctionAdapter() {
   public Object doNew(JSObject thisObject, Object[] args)
         throws JSException {
      JSGlobalObject go = thisObject.getGlobalObject();
      JSObject proto = go.makeJSObject();

      proto.setMember("log", new JSFunctionAdapter() {
         public Object doCall(JSObject thisObject, Object[] args)
               throws JSException {
            if (args.length >= 1) {
               Extension.log(args[0].toString());
            }

            return null;
         }
      });
      final JSObject obj = go.makeJSObject(proto);
      return obj;
   }
});
```

With this constructor, scripts can do the following:

```
var logger = new Logger();
logger.log("message");
```

Conversion to a script extension

When you convert a FESI extension to a script extension, the root of a script extension is the `ScriptExtension` interface.

You must implement this interface to create script extension.

```
public class ITIMExtension implements ScriptExtension {

   public List getContextItems() {
   }

   public void initialize(ScriptInterface si, ScriptContextDAO dao)
         throws ScriptException, IllegalArgumentException {
   }
}
```

To create object that can be used in scripts, create a POJO class that contains all of the business logic, and implements the marker interface `ExtensionBean`. A marker interface means that `ExtensionBean` does not require you to implement any methods and it does add any new data to your class. A POJO that implements `ExtensionBean` is treated specially by the IBM Security Identity Manager scripting components.

If your class does not implement `ExtensionBean`, then scripts cannot use the methods provided by the POJO. For example:

```
public class Extension implements ExtensionBean {
   public static void log(String msg) {
      System.out.println(msg);
   }
}
```

In the initialize method of your extension, create `ContextItem` that contains an instance of your extension and add that `ContextItem` to a List.

```
ContextItem custom = ContextItem.createItem("CustomExtension",
      new Extension());
items.add(custom);
```

To create global function, use `ContextItem`, but this time call `createGlobalFunction`. For example:

```
ContextItem func = ContextItem.createGlobalFunction("log",
      new GlobalFunction() {
         public Object call(Object[] parameters)
               throws ScriptEvaluationException {
            if (parameters.length >= 1) {
               Extension.log(parameters[0].toString());
            }
            return null;
         }
      });
items.add(func);
```

The second argument to `createGlobalFunction` is a `GlobalFunction` object. `GlobalFunction` has a single method that you must implement and call. It is similar to the `doCall` method from the FESI `JSFunctionAdapter`. `GlobalFunctions` are not suggested because, like the `doCall` method, they pass an array of parameters. You must check that all of the parameters exist and are the right types, which can be difficult to maintain over the life of your extension.

Creation of a constructor

To create a constructor, use `ContextItem` and the `createConstructor` method.

For example:

```
ContextItem logger = ContextItem.createConstructor("Logger",
      Extension.class);
items.add(logger);
```

The second parameter to `createConstructor` is the Class object for the object that you want to construct. It is usually a POJO that implements `ExtensionBean`.

In each of these examples, you add the `ContextItem` to a List. In the `getContextItems` method of `ScriptExtension`, you return that List. For example, the full code is:

```
public class ITIMExtension implements ScriptExtension {

   private List<ContextItem> items;

   public List getContextItems() {
      return items;
   }

   public void initialize(ScriptInterface si, ScriptContextDAO dao)
         throws ScriptException, IllegalArgumentException {

      items = new ArrayList<ContextItem>();

      ContextItem custom = ContextItem.createItem("CustomExtension",
            new Extension());
      items.add(custom);

      ContextItem func = ContextItem.createGlobalFunction("log",
            new GlobalFunction() {
               public Object call(Object[] parameters)
                     throws ScriptEvaluationException {
                  if (parameters.length >= 1) {
```

```
                    Extension.log(parameters[0].toString());
                }
                return null;
            }
        });
    items.add(func);

    ContextItem logger = ContextItem.createConstructor("Logger",
        Extension.class);
    items.add(logger);
  }
}
```

Chapter 10. JavaScript extension reference

The reference section is arranged alphabetically.

There are a number of IBM Security Identity Manager specific objects available for use. IBM Security Identity Manager uses JavaScript extensions to package JavaScript objects and APIs. An extension can also be a package of other extensions (for example, `ModelExtension`).

After an extension is defined, it can be registered in the `scriptframework.properties` file to be used in a specific JavaScript context. Use the Update Property page from the **Appliance Dashboard** of the IBM Security Identity Manager virtual appliance console. See Managing the server properties. In some cases, an environment needs to be created for an extension.

Table 9 shows these script extensions.

Table 9. Script extensions

Script Extension	Object Name	Object Type
AttributesExtension (deprecated)	**ATTRIBUTES**	**Map**
EmailContextExtension	**EmailContext**	**EmailContext**
EnroleExtension	**Enrole** **error**	**Enrole** **Error**
IdentityPolicyExtension	**IdentityPolicy**	**IdentityPolicy**
LoopCountExtension	**loopcount**	**int**
PersonPlacementRulesExtension	**entry**	**Map**
PostOfficeExtension	**PostOffice**	**PostOffice**
ProvisioningPolicyExtension	**parameters** **reason**	**Map** **int (0: New Account, 1: Existing Account)**
AccountModelExtension	**Account constructor** **AccountSearch constructor**	**Account** **AccountSearch**
OrganizationModelExtension	**ContainerSearch constructor**	**ContainerSearch**
PersonModelExtension	**Person constructor** **ExtendedPerson constructor** **PersonSearch constructor**	**Person** **ExtendedPerson** **PersonSearch**
RoleModelExtension	**Role constructor** **RoleSearch constructor**	**Role** **RoleSearch**
ServiceModelExtension	**Service constructor** **ServiceSearch**	**Service** **ServiceSearch**
ReminderExtension	**reminderCtx**	**Reminder**
ServiceExtension	**service**	**DirectoryObject**

Table 9. Script extensions (continued)

Script Extension	Object Name	Object Type
SubjectExtension	**subject**	**Person** **Note:** For Orphan Adoption Rule JavaScript, the subject is a Map, which contains the account attributes returned from reconciliation. The entries in the map are referred by the name of the account attributes, which might vary based on the service type.
WorkflowExtension	**process** **activity** **Participant constructor** **ParticipantType** **$RelevantDataName**	**Activity** **Participant** **ParticipantType** **ProcessData**Process

Finding methods and properties for a specific JavaScript object

This example demonstrates how to find methods and properties for a specific JavaScript object.

If you are writing a workflow script, look in the `scriptframework.properties` file to see which extensions are available. By default, workflow loads the model extensions, the `WorkflowExtension`, and the `LoopCountExtension`.

Table 9 on page 75shows that `WorkflowExtension` defines scripting objects that include process, activity, a Participant constructor, an object named `ParticipantType`, and a series of workflow-specific pieces of data.

In another column in the table, notice that the process object is of type Process. Now, locate Process in this reference to see that Process type has a property called name, and a method called `getParent()`.

To understand how to use maps, notice that objects, such as `parameters` from **`ProvisioningPolicyExtension`**, have a type of Map. A Map, also known as a dictionary, is a named JavaScript object that can hold many other objects which can be accessed by name. The `parameters` object holds another object named `uid`. To access `uid`, you can type `parameters.uid[0]`. (In this case `uid` is an array, so you must type `[0]` to get the first element of the array.) The values that a map holds will vary between each map. For more information, locate the specific map in the JavaScript reference.

How to read the reference pages

This section explains the structure of each reference item.

Title and Description
Every reference entry begins with a title and a one line description. The entries are alphabetized by title. The one-line description gives a quick summary of the item documented in the entry.

Availability
The IBM Security Identity Manager JavaScript extensions change over time. Unless otherwise noted, anything available in one version of the IBM Security Identity Manager extensions is also available in later versions. This section also specifies whether an existing item was enhanced with a

later version of the extensions and when an item is deprecated. Deprecated items are no longer supported and can be removed from future versions of the IBM Security Identity Manager extensions. Do not use deprecated items in new IBM Security Identity Manager JavaScript code.

Provided by

At installation, IBM Security Identity Manager provides this initial set of registered extensions:

- **`EnroleExtension`**
- **`ProvisioningPolicyExtension`**
- **`PostOfficeExtension`**
- **`IdentityPolicyExtension`**
- **`PersonPlacementRulesExtension`**
- **`WorkflowExtension`**
- **`ReminderExtension`**
- **`ServiceExtension`**
- **`SubjectExtension`**
- **`AttributesExtension`**
- **`LoopCountExtension`**
- **`EmailContextExtension`**
- **`Model extensions package`**

Inherits From

JavaScript classes can inherit properties and methods from other classes. When it occurs, an Inherits From section appears in the reference entry. The inherited fields and methods are in the listed superclasses. For example, the subject object inherits all of its fields and properties from the **`DirectoryObject`** class.

Synopsis

This section is a synopsis of how to use the object, method, property, or function.

Arguments

If the reference page describes a function or method that has arguments, the Synopsis is followed by an Arguments subsection that describes the arguments to the function or method. For some objects, the Synopsis section is replaced by a Constructor section which is also followed by an Arguments subsection.

Returns

If a function or a method has a return value, the Arguments subsection is followed by a Returns subsection that explains the return value of the function, method or constructor.

Properties

If the reference page documents an object, the Properties section lists the properties the object supports and provides short explanations of each.

Methods

The reference page for an object that defines methods includes a Methods section.

Description

Most reference entities contain a Description section, which is a basic description of whatever is documented. For some simple methods, the

Arguments and Returns sections document the method sufficiently by themselves, so the Description section is omitted.

Usage This section describes common techniques for using the item, or it contains cautionary information.

Account

Represents an account that is associated with a provisioning operation.

Availability
IBM Security Identity Manager 7.0.

Inherits From
`DirectoryObject`

Provided by
`com.ibm.itim.script.extensions.model.AccountModelExtension`

Constructor
`new Account(dn)`

Returns
The newly created Account object that represents the account with the specified DN, which is a String.

Methods
`getAndDecryptPassword()`

Decrypts and returns
The decrypted password of the account entity in plain text.

Note: This method is available in the scripting context of Security Identity Manager only if the javascript.password.access.enabled property is set to `true` in the `scriptframework.properties` file.

`setAndEncryptPassword()`

Encrypts
The given plaintext password and sets it on the account object.

Note: This method is available in the scripting context of Security Identity Manager only if the javascript.password.access.enabled property is set to `true` in the `scriptframework.properties` file.

Account.getAndDecryptPassword()

The method decrypts and returns the decrypted password of the account entity in plain text.

Availability
IBM Security Identity Manager 7.0.

Synopsis
`account.getAndDecryptPassword()`

Returns
String representing plain text password set in the account object.

Description

This method can be used in the scripting context of Security Identity Manager if the **`javascript.password.access.enabled`** property is set to

`true` in the `scriptframework.properties` file. It decrypts and returns the decrypted password set in the account object. This function will return null if the password is not present.

Note: This method does not decrypt the password of the Security Identity Manager account, which is hashed and stored in LDAP.

Usage

```
var password = account.getAndDecryptPassword();
</page_ Account.getAndDecryptPassword()>
```

Account.setAndEncryptPassword()

The method encrypts the given plaintext password and sets it on account object.

Availability

IBM Security Identity Manager 7.0.

Synopsis

```
account.setAndEncryptPassword(String password)
```

Arguments

password
Plain text password string.

Description

This method can be used in the scripting context to set a given plain text password to an account object if the **`javascript.password.access.enabled`** property is set to `true` in the `scriptframework.properties` file. Internally, the function encrypts the password and sets the same on the account entity.

Usage

```
account.setAndEncryptPassword("secret");
</page_ Account.setAndEncryptPassword()>
```

AccountSearch

You can search for an account with the `AccountSerach` object.

Availability

IBM Security Identity Manager 7.0.

Provided by

`com.ibm.itim.script.extensions.model.AccountModelExtension`

Constructor

```
new AccountSearch()
```

Returns

The newly created and initialized account search object.

Methods

searchByOwner()
Search for an account by owner.

searchByUid()
Search for an account by user ID.

searchByUidAndService()
Search for an account by user ID and service.

searchByURI()
Search for an account by URI within an organizational container.

Description
The entity implements the IBM Security Identity Manager Account Search class.

AccountSearch.searchByOwner()

The method finds an account entity by the distinguished name of the owner.

Availability
IBM Security Identity Manager 7.0.

Synopsis
```
AccountSearch.searchByOwner(personDN)
```

Arguments

personDN
String representing the distinguished name of the account owner.

Description
Given the distinguished name of the person, find the account entities owned by that person. This function will return `null` if the person is not found.

Usage
```
var account = (new AccountSearch()).searchByOwner(person.dn);
if (account!=null) {
Enrole.log("script", "Found " + account.length + " accounts");}
```

AccountSearch.searchByUid()

The method finds an account entity by user ID and distinguished name of a service.

Availability
IBM Security Identity Manager 7.0.

Synopsis
```
AccountSearch.searchByUid(uid, serviceDN)
```

Arguments

uid String representing the user ID of the account.

serviceDN
String representing the distinguished name of the account.

Description
Given the user ID of the account and the distinguished name of the service, find the account entity. This function returns `null` if there is not exactly one matching account, or if the service is not found.

Usage
```
var account = (new AccountSearch()).searchByUid("pallen",
service.dn);
if (account!=null) {
Enrole.log("script", "Found account pallen");
}
```

AccountSearch.searchByUidAndService()

The method finds an account entity by user ID, service name, and service profile name.

Availability

IBM Security Identity Manager 7.0.

Synopsis

```
AccountSearch.searchByUidAndService(uid, serviceName)
```

Arguments

uid String representing the user ID of the account.

serviceName
String representing the name of the service.

Description

Given the user ID of the account and the name of the service that has the same service profile as the script context service profile, find the account entity. This function returns `null` if:

- More than one matching account exists.
- The service is not found.
- More than one service with the given name exists.

Usage

```
var account = (new AccountSearch()).searchByUidAndService
 ("pallen", "Domain Controller");
if (account!=null) {
 Enrole.log("script", "Found account pallen"); }
```

Synopsis

```
AccountSearch.searchByUidAndService(uid, serviceName,
serviceProfileName)
```

Arguments

uid String representing the user ID of the account.

serviceName
String representing the name of the service.

serviceProfileName
String representing the name of the service profile of the serviceName service.

AccountSearch.searchByURI()

The method finds an account by URI in an organizational container.

Availability

IBM Security Identity Manager 7.0

Synopsis

```
AccountSearch.searchByURI(containerDN, uri)
```

Arguments

Container DN
String representing the distinguished name of the organizational container.

uri String representing the URI of the account.

Returns

An Account object.

Description

Given the distinguished name of an organizational container and the account URI, this method finds the account. If the account is not found, this function returns null. If more than one account is found, this function throws a scripting exception.

Usage

```
var account = (new AccountSearch()).searchByURI(container.dn, uri);
if (account != null) {
Enrole.log("script", "Found " + account.getProperty("eruid") );}
```

Activity

Activity is used to reference any activity in a IBM Security Identity Manager workflow.

Availability

IBM Security Identity Manager 7.0

Provided by

The activity JavaScript object in the `WorkflowExtension` returns an Activity object that represents the current workflow activity. The workflow activity can be used in the context of a workflow activity PostScript, or in a transition script, to reference the current activity. For a transition script, this object represents the activity whose completion has lead to the evaluation of the transition script.

`Process.getActivity()` can return any Activity object in the context of a workflow process. For more information, see the description of this method.

Activity Result Summary Code

APPROVED

Approved process summary code. Result code is `AA`.

ESCALATED

Escalated process summary code. Result code is `ES`.

FAILED

Failed process summary code. Result code is `SF`.

PARTICIPANT_RESOLVE_FAILED

Participant resolved failure process summary code. Result code is `PF`.

PENDING

Pending process summary code. Result code is `PE`.

REJECTED

Rejected process summary code. Result code is `AR`.

SUBMITTED

Submitted process summary code. Result code is `RS`.

SUCCESS

Success process summary code. Result code is `SS`.

TIMEOUT

Time out process summary code. Result code is `ST`.

WARNING
Warning process summary code. Result code is `SW`.

Properties

description
Describes the purpose of the activity given when defined in the workflow designer.

duedate
Indicates the time in milliseconds by when the activity is due.

id Assigned by the workflow designer to uniquely identify the workflow activity within the workflow engine.

index Index of the instance of the activity.

name Label given this activity when defined in the workflow designer.

participant
The activity participant, as defined in the workflow designer.

resultDetail
An application-specific string that provides more detail about the result of the activity.

resultSummary
An application-specific string that represents the summary result of the activity.

started
Indicates when the activity started.

state Code that represents the current state of the activity.

subtype
Code that further categorizes the activity beyond the type of the activity, such as approval or request for information.

type Code that categorizes the activity given when defined in the workflow designer, such as manual or application.

Methods

auditEvent()
Create an event in the audit trail specific to the activity.

setResult()
Change the result member of the activity in the current activity.

Description
This entity represents the current workflow activity that is being run. Within the context of a workflow transition script, this entity represents the activity whose completion has lead to the evaluation of the transition script. No constructor is available to create this object in any IBM Security Identity Manager context.

Activity.auditEvent()

The method creates an event in the audit trail.

Availability
IBM Security Identity Manager 7.0.

Synopsis
activity`.auditEvent(event)`

Arguments

event String representing the event to be audited.

Description

This method creates an event in the audit trail specific to the activity. The function takes in one parameter that can be any JavaScript object that can be translated into a String for storage. In the audit trail, the event is automatically time stamped.

Usage

```
activity.auditEvent("Task completed");
```

Activity.description

The field provides information about the purpose of the activity.

Availability

IBM Security Identity Manager 7.0

Synopsis

activity.description

Description

This read-only field is a String that describes the purpose of the activity given when defined in the workflow designer.

Usage

```
x = activity.description;
```

Activity.duedate

The field represents the time in milliseconds by when the activity is due.

Availability

IBM Security Identity Manager 7.0.

Synopsis

activity.duedate

Description

This read-only field is a long number of milliseconds by when this activity is due.

Usage

```
x = activity.duedate;
```

Activity.getSubProcesses()

The method returns the subordinate processes (if any) of the activity.

Availability

IBM Security Identity Manager 7.0.

Synopsis

```
activity.getSubProcesses()
```

Returns

The subordinate processes. If there are no subordinate processes, an empty array is returned.

Description

This method returns the subordinate processes (if any) of this activity.

Usage

```
var out = "subprocesses of the activity: \n";

var subProcesses = activity.getSubProcesses();
for (var i = 0; i < subProcesses.length; i++) {
  out += subProcesses[i].id + " type: " + subProcesses[i].type + " resultSummary: " + subProcesses[i].resultSummary + "\n";
}

activity.auditEvent(out);
```

Activity.guid

The generated unique identifier assigned to the activity at runtime.

Availability
IBM Security Identity Manager 7.0

Synopsis
activity.guid

Description
This read-only field is a String of the generated unique identifier for the workflow activity within the workflow engine.

Usage `x = activity.guid;`

Activity.id

The field is the unique identifier assigned to the activity.

Availability
IBM Security Identity Manager 7.0.

Synopsis
activity.id

Description
This read-only field is a String assigned by the workflow designer to uniquely identify the workflow activity within the workflow engine.

Usage `x = activity.id;`

Activity.index

The field is an index of the instance of the activity.

Availability
IBM Security Identity Manager 7.0.

Synopsis
activity.index

Description
This field is a read-only and a number. If there is more than one instance of this activity, such as in the case where the activity of the ID is called multiple times in a loop in the workflow process, the value starts at one. If there is only one instance of this activity, the index value is zero.

Usage `x = activity.index;`

Activity.name

The field is the label that is assigned to the activity.

Availability
IBM Security Identity Manager 7.0

Synopsis
activity.name

Description
This read-only field is a String assigned by the workflow designer to label this activity.

Usage `x = activity.name;`

Activity.participant

The field represents the activity participant.

Availability
IBM Security Identity Manager 7.0

Synopsis
activity.participant

Description
This read-only field is a Participant that represents the activity participant. Not all activities have a participant. If there is no participant associated with the activity, this member is empty.

Usage `x = activity.participant;`

Activity.resultDetail

You can get the details about the result of the activity with this field.

Availability
IBM Security Identity Manager 7.0.

Synopsis
activity.resultDetail

Description
This read-only field is an application-specific string that provides more detail about the result of the activity.

Usage `x = activity.resultDetail;`

Activity.resultSummary

The field helps you view the summary of the result of the activity.

Availability
IBM Security Identity Manager 7.0.

Synopsis
activity.resultSummary

Summary
This read-only field is an application-specific string that provides a summary of the result of the activity. It can represent a success or failure.

Usage `x = activity.resultSummary;`

Activity.setResult()

The method changes the result member of the activity.

Availability
IBM Security Identity Manager 7.0.

Synopsis

```
activity.setResult(summary)
activity.setResult(summary, detail)
```

Arguments

summary
String code that represents the result summary.

detail String representing the result details.

Description
This method changes the result member of the activity in the current activity. It is supported for current activities in the current workflow process. The result is composed by an application-specific summary code, and optional more detailed application-specific description. The summary code can indicate a success or failure. This summary code is stored as the `resultSummary` member locally and updated in the relevant data in the workflow engine. The detail is stored as the `resultDetail` member locally and updated in the relevant data in the workflow engine.

Usage

```
activity.setResult(activity.FAILED);
activity.setResult(activity.FAILED, "Unable to connect to resource");
```

Activity.started

The field represents the date that indicates when the activity started.

Availability
IBM Security Identity Manager 7.0.

Synopsis

```
activity.started
```

Description
This read-only field is a string that represents the date that indicates when the activity started.

Usage

```
x = activity.started;
```

Activity.state

The field represents the current state of the activity.

Availability
IBM Security Identity Manager 7.0

Synopsis

```
activity.state
```

Description
This read-only field is a code string that represents the current state of the activity. The state can have the following values:

- **R** for running
- **I** for not started
- **T** for terminated
- **A** for aborted

- **S** for suspended
- **C** for completed
- **B** for bypassed

Usage

```
if (activity.state == "S") {
  ...
}
```

Activity.subtype

The field represents the subtype of the activity.

Availability
IBM Security Identity Manager 7.0.

Synopsis
activity.subtype

Description
This read-only field is a code string that further categorizes the activity beyond the type of the activity, such as approval or request for information. This is defined in the workflow designer. Not all activities have a subtype. If there is no subtype associated with the activity, this member is empty. The currently supported subtypes are:

- **AP** for approval
- **RI** for request for input
- **WO** for work order

Usage `x = activity.subtype;`

Activity.type

The field represents the type of the activity.

Availability
IBM Security Identity Manager 7.0.

Synopsis
activity.type

Description
This read-only field is code string that categorizes the activity given when defined in the workflow designer, such as manual or application. The currently supported types are:

- **S** for subprocess
- **L** for loop
- **A** for application
- **R** for route
- **M** for manual
- **O** for operation

Usage `x = activity.type;`

AttributeChangeOperation

The object represents an entity about the attribute change operation.

Availability
IBM Security Identity Manager 7.0.

Provided by
AttributeChangeOperation objects are returned from the method **DirectoryObject.getChanges()** and are therefore not provided by any specific extension.

Properties

attr Name of the attribute that is being changed.

op An integer that identifies the type of change that is being made.

values[]
An array of objects that must be either added, removed, or replaced.

Description
This entity represents the changes made to a IBM Security Identity Manager object.

AttributeChangeOperation.attr

Represents the name of an attribute that is being changed.

Availability
IBM Security Identity Manager 7.0.

Synopsis
attributeChangeOperation.attr

Description
Value is the attribute that is being changed.

Usage
```
x = attributeChangeOperation.attr;
```

AttributeChangeOperation.op

The field represents the type of change that is being made.

Availability
IBM Security Identity Manager 7.0.

Synopsis
attributeChangeOperation.op

Description
This read-only field is a number that identifies the type of change that is being made. The values are:

- **1** for add
- **2** for replace
- **3** for remove

Usage
```
x = attributeChangeOperation.op;
```

AttributeChangeOperation.values[]

The field represents the name of attribute that is being changed.

Availability
IBM Security Identity Manager 7.0.

Synopsis
attributeChangeOperation.values[]

Description
This read-only field is an array of objects that must be added, removed, or replaced.

Usage `x = attributeChangeOperation.values[1];`

ContainerSearch

The object represents the search for an organizational container.

Availability

IBM Security Identity Manager 7.0.

Provided by
`com.ibm.itim.script.extensions.model.OrganizationModelExtension`

Constructor
`new ContainerSearch()`

Returns
The newly created and initialized container search object.

Methods

searchByFilter()
Search for a container with a filter.

searchByURI()
Search for an organizational container by URI within a parent organizational container.

Description
Implements the IBM Security Identity Manager **`OrganizationalContainerSearch`** class.

ContainerSearch.searchByFilter()

The method represents the search for a container with a filter.

Availability
IBM Security Identity Manager 7.0.

Synopsis
`containerSearch``.searchByFilter(`*`profileName`*`, `*`filter`*`, `*`scope`*`)`

Arguments

profileName
The String name of the organizational container profile to use.

filter LDAP search filter String that defines the criteria for returned containers to meet. The filter must be in the format defined by RFC2254.

scope Optional Int search scope. Use 1 for One Level Scope and 2 for SubTree Scope. One Level Scope is the default scope.

Returns
An array of **`DirectoryObjects`** representing the results of the search.

Description
This method searches for a container with a filter.

Usage

```
var locationContainer = new ContainerSearch();
// use subtree scope
var thisLocation = locationContainer.searchByFilter("Location",
 "(l=Raleigh)", 2);

// use default one level scope
var otherLocation = locationContainer.searchByFilter("Location",
 "(l=Raleigh)");
```

ContainerSearch.searchByURI()

The method finds an organizational container by URI in a parent organizational container.

Availability

IBM Security Identity Manager 7.0.

Synopsis

```
ContainerSearch.searchByURI(containerDN, uri)
```

Arguments

Container DN
String representing the distinguished name of the parent organizational container.

uri String representing the URI of the organizational container.

Returns

A **DirectoryObject** representing the container.

Description

Given the distinguished name of the parent organizational container and the container URI, this method finds the container. If the container is not found, this function returns null. If more than one container is found, this function throws a scripting exception.

Usage

```
var container = (new ContainerSearch()).searchByURI(parentContainer.dn,
   uri);
if (container != null) {
Enrole.log("script", "Found " + container.getProperty("ou") );}
```

Context

The object represents the context of the currently running workflow process (for example, requestor or subject). Only used for entitlement workflows.

Note: This object type is deprecated. Use workflow JavaScript objects, such as **Process**, **Activity**, and **Relevant Data**.

Some account-specific functions of the context JavaScript extension, including **getService()**, **isAccountDataChanged()**, and **getAccountParameter()** cannot be applicable to operation workflows that are not account related. The context JavaScript extension is not suggested for custom workflows.

Availability

IBM Security Identity Manager 7.0.

Provided by

`com.ibm.itim.workflow.script.WorkflowExtension`

Context Constants

APPROVED

This constant is used to describe the result of an activity. The member applies only to Approval types of activities.

Usage

```
if (context.getActivityResult() == context.APPROVED) {...
```

REJECTED

This constant is used to describe the result of an activity. This member applies only to Approval types of activities.

Usage

```
if (context.getActivityResult() == context.REJECTED) {...
```

NEWACCOUNT

This constant is used to identify the type of request that triggers the custom workflow run time.

Usage

```
if (context.getProcessType() ==
  context.NEWACCOUNT) {...
```

ACCOUNTDATACHANGE

This constant is used to identify the type of request that triggers the custom workflow in run time.

Usage

```
if (context.getProcessType() ==
  context.ACCOUNTDATACHANGE) {...
```

Methods

getAccountParameter()

Returns the value of an account attribute.

getActivityResult()

Returns the activity result for the current activity.

getActivityResultByID()

Returns the activity result for a specific activity.

getLoopCount()

Returns the loop count for the current loop activity.

getLoopCountByID()

Returns the current loop count for a specific loop activity.

getProcessType()

Returns the type of the request that triggers the custom workflow process.

getRequestee()

Returns the requestee associated with the request as a Person object.

getService()

Returns the target service as a Service entity object.

isAccountDataChanged()

Identifies whether a specific account attribute was changed in the request that triggers the custom workflow process.

Description

The context of the currently running workflow process (for example, requestor or subject) is represented within the JavaScript as an object named context.

Context.getAccountParameter()

The method returns the value of an account attribute.

Availability
IBM Security Identity Manager 7.0.

Synopsis
`context.getAccountParameter(String attributeName)`

Arguments

attributeName
String representing the attribute name.

Returns
String value of an account attribute.

Description
This member function returns the value of an account attribute as a string.

Usage
```
parameter=context.getAccountParameter("group");
```

Context.getActivityResult()

The method returns the activity result for the current activity.

Availability
IBM Security Identity Manager 7.0.

Synopsis
`context.getActivityResult()`

Returns
String

Description
This member function returns the activity result for the current activity. The function returns APPROVED or REJECTED. If this function is used to specify a transition condition, the function refers to the activity from which the transition is coming.

Usage
```
if (context.getActivityResult() == context.APPROVED) {...
```

Context.getActivityResultById()

The method returns the activity result for a specific activity.

Availability
IBM Security Identity Manager 7.0.

Synopsis
`context.getActivityResultById(String activityDefinitionID)`

Arguments

activityDefinitionID
String ID of the activity definition.

Returns
String

Description
This member function returns the activity result for a specific activity. The function returns APPROVED or REJECTED.

Usage `if (context.getActivityResultByID("1234567890") == context.APPROVED) {...`

Context.getLoopCount()

The method returns the loop count for the current loop activity.

Availability
IBM Security Identity Manager 7.0.

Synopsis
`getLoopCount()`

Returns
Integer of loop count.

Description
This member function returns the loop count for the current loop activity. If this function is called before a loop is started, the loop count is 0. If this activity is called while the loop activity is in process, the loop count is the number of times the loop ran. If this function is called after the loop is completed, the loop count is the total number of times the loop is defined to run.

Usage `currentiteration = context.getLoopCount();`

Context.getLoopCountByID()

The method returns the current loop count for a specific loop activity.

Availability
IBM Security Identity Manager 7.0.

Synopsis
`context.getLoopCountByID(`*`String activityDefinitionID`*`)`

Arguments

activityDefinitionID
ID of the activity definition.

Returns
Integer

Description
This member function returns the current loop count for a specific loop activity. If this function is called before the loop is started, the loop count is 0. If this function is called while the loop activity is in process, the loop count is the number of times the loop ran. If this function is called after the loop is completed, the loop count is the total number of times the loop is defined to run.

Usage `currentiteration = context.getLoopCount("1234567890");`

Context.getProcessType()

The method returns the type of the request that triggers the custom workflow process.

Availability
IBM Security Identity Manager 7.0.

Synopsis
`context.getProcessType()`

Returns
String

Description
This member function returns the type of the request that triggers the custom workflow process. The function returns `NEWACCOUNT` or `ACCOUNTDATACHANGE`.

Usage
```
if (context.getProcessType() == context.NEWACCOUNT) {...
```

Context.getRequestee()

The method returns the requestee associated with the request as a person object.

Availability
IBM Security Identity Manager 7.0.

Synopsis
```
context.getRequestee();
```

Returns
A `DirectoryObject` that represents a Person.

Description
This member function returns the requestee associated with the request as a Person object. The requestee is the user who owns the associated, provisioned account.

Usage
```
requestee = context.getRequestee();
```

Context.getService()

The method returns the target service as a service entity object.

Availability
IBM Security Identity Manager 7.0.

Synopsis
```
context.getService()
```

Returns
`DirectoryObject`

Description
This member function returns the target service as a Service entity object. The service entity is the service associated with the provisioned account.

Usage
```
service = context.getService();
```

Context.isAccountDataChanged()

The method identifies whether a specific account attribute was changed in the request that triggers the custom workflow process.

Availability
IBM Security Identity Manager 7.0.

Synopsis
`isAccountDataChanged(`*`String attributeName`*`)`

Description
This member function identifies whether a specific account attribute was changed in the request that triggers the custom workflow process. If the request that triggers the custom workflow is `NEWACCOUNT` and the attribute is in the new account parameters, this function returns TRUE. Otherwise,

this function returns FALSE. If the request that triggers the custom workflow is ACCOUNTDATACHANGE and the specified attribute is changed, this function returns TRUE. Otherwise, this function returns FALSE.

Usage `if (context.isAccountDataChanged("group")) {...`

Delegate

The object provides the Delegate JavaScript object for use in the JavaScript environment of delegation notification. The Delegate JavaScript object and their use is described in this section.

Delegate

The Delegate object contains all the information associated with the current delegation operation.

Availability

IBM Security Identity Manager 7.0

Delegation Notification context

Provided by

com.ibm.itim.script.extensions.DelegateExtension

Methods

Delegate.getDelegator()

Returns the DirectoryObject that represents a system user such as the IBM Security Identity Manager account, whose activities are delegated.

Delegate.getDelegatee()

Returns the DirectoryObject that represents a system user such as the IBM Security Identity Manager account, who is selected to be the delegate for the activities of the delegator.

Delegate.getStartDate()

Returns a Date that contains the date and time when the delegation starts.

Delegate.getEndDate()

Returns a Date that contains the date and time when the delegation ends.

Delegate.getRequester()

Returns the DirectoryObject that represents a system user such as the IBM Security Identity Manager account, who initiated the delegation.

Description

The Delegate object is available in the context of a delegation notification. The object retrieves the delegation information in the delegation notification template. The model script extensions are also available in the delegation notification context.

DirectoryObject

The object represents any IBM Security Identity Manager directory object or entity.

Availability

IBM Security Identity Manager 7.0

Constructor

There is no specific constructor for this object. Specific constructors for Account, Person, Role, and Service return `DirectoryObject`.

For example, `new Service()` returns a `DirectoryObject`.

Properties

dn String representing the distinguished name of the entity.

name String representing the logical name of the entity.

profileName
String representing the profile name of the entity.

Methods

addProperty()
Changes the value of the specified property, or adds the specified property if it does not exist. For multivalued objects, `addProperty()` adds the values to the specified property in the directory object and does not replace them.

getChanges()
Returns the changes made to the entity.

getProperty()
Returns the values of the property specified by the given name.

getPropertyNames()
Returns a list of properties (attributes and relationships).

removeProperty()
Removes the specified property.

setProperty()
Changes the value of the specified property, or adds the specified property if it does not exist.

getPropertyAsDate()
Returns the value of the specified property as a Date.

getPropertyAsString()
Returns the value of the specified property as a String.

Description

This Object represents a Security Identity Manager entity in the JavaScript environment. Each Security Identity Manager entity is wrapped in one of these object classes.

DirectoryObject.addProperty()

The method adds or updates the value for the specified property.

Availability

IBM Security Identity Manager 7.0

Synopsis

directoryObject.addProperty(*name*, *value*)

Arguments

name String representing the name of the property to be created or modified.

value The value to add to the property.

Description

This method changes the value of the specified property or adds the specified property if it does not exist. This change is made locally to the script environment, not to the data store. The value can be a single value object or an array of objects. For multivalued objects, `addProperty()` adds the values to the specified property in the directory object and does not replace them. The value type (syntax) of object must be compatible with the syntax of the specified property. This method is available for the following data types:

- `void addProperty(String name, Collection value);`
- `void addProperty(String name, Date value);`
- `void addProperty(String name, Map value);`
- `void addProperty(String name, boolean value);`
- `void addProperty(String name, byte value);`
- `void addProperty(String name, String value);`
- `void addProperty(String name, number value);`
- `void addProperty(String name, char value);`

Usage

```
directoryObject.addProperty("eruid", "jdoe");
```

The `getProperty` method returns a Java array of objects that is stored in a JavaScript `JavaArray` object. Unlike a standard JavaScript array, `JavaArray` objects are used to access members of a Java array. Because Java arrays cannot be resized, the size of a `JavaArray` object cannot be changed. Also, `JavaArray` objects are typed. Setting a `JavaArray` element to the wrong type throws a JavaScript error.

In Security Identity Manager, a `JavaArray` object cannot be passed directly back into a `addProperty` method. The `JavaArray` array might be converted into a standard JavaScript array as follows:

```
jsAliases = new Array();
myPerson = person.get();
aliases = myPerson.getProperty("eraliases");
for (i=0; i < aliases.length; i++) {
jsAliases[i] = aliases[i];
}
jsAliases[aliases.length] = "myNewAlias";
myPerson.addProperty("eraliases", jsAliases);
person.set(myPerson);
```

DirectoryObject.dn

The field represents the distinguished name of the object.

Availability

IBM Security Identity Manager 7.0.

Synopsis

directoryObject.dn

Description

This read-only field is a string that provides the distinguished name of the object. If the object holds information that was not created, there is no value.

Usage `x = directoryObject.dn;`

DirectoryObject.getChanges()

The method returns the changes made to the entity.

Availability

IBM Security Identity Manager 7.0.

Synopsis

directoryObject.getChanges()

Returns

An array of change objects. If there are no changes, an empty array is returned. Each element in the array is an AttributeChangeOperation.

Description

This method returns the changes made to the entity. These changes are represented by change objects with the following members:

attr String name of the attribute that is being changed.

op An integer that identifies the type of change that is being made. The enumerated values are 1 for add, 2 for replace, and 3 for remove.

values An array of objects that must be either added, removed, or replaced.

The changes are returned as an array of these change objects. If there are no changes, an empty array is returned.

Usage

```
changes = directoryObject.getChanges();
for (i = 0; i < changes.length; i++) {
  name = changes[i].attr;
  if (changes[i].op == 1) {
    ...
  } else if (changes[i].op == 2) {
    ...
  } else {
    ...
  }
};
```

DirectoryObject.getProperty()

The method returns the values of the property specified by the given name.

Availability

IBM Security Identity Manager 7.0

Synopsis

directoryObject.getProperty(*name*)

Arguments

name String representing the name of the property to return.

Returns

Either a String or a DirectoryObject. The type of object returned depends on the property obtained. If the specified property does not exist, an empty array is returned.

Description

This method returns the values of the property specified by the given

name. The type of object returned depends on the property obtained. If the specified property does not exist, an empty array is returned.

The property name can be either an attribute name or a relationship name. For an attribute name, the return is a `String[]`; for a relationship name, an array of `DirectoryObjects` is returned. If an attribute and a relationship have the same name, then the attribute is returned. For example, an Account entity has both an owner attribute and an owner relationship.

Usage When operating on an account, for example, the user ID property can return a String, where the owner property can return another entity (`DirectoryObject`). The owner entity can then be operated on with the `getProperty()` member to obtain information about it.

```
userids = directoryObject.getProperty("eruid");
if (userids.length > 0)
    userid = userids[0];
owner = directoryObject.getProperty("owner");
if (owner.length > 0)
    ownerName = owner.getProperty("name")[0];
```

Note: These statements assume there is at least one value returned. If no values are returned, an array indexing violation occurs.

The `getProperty` method returns a Java array of objects that is stored in a JavaScript `JavaArray` object. Unlike a standard JavaScript array, `JavaArray` objects are used to access members of a Java array. Since Java arrays cannot be resized, the size of a `JavaArray` object cannot be changed. Also, `JavaArray` objects are typed. Setting a `JavaArray` element to the wrong type throws a JavaScript error.

DirectoryObject.getPropertyAsDate()

The method returns the value of the property specified by the given name as a date object.

Availability
IBM Security Identity Manager 7.0.

Synopsis
directoryObject`.getPropertyAsDate(name)`

Arguments

name String representing the name of the property to return.

Returns
A `Date` object. If the specified property does not exist, current date is returned.

Description
This method returns the value of the property specified by the given name as a date object. If the specified property does not exist, current date is returned.

Usage

```
var createDate = directotyObject. getPropertyAsDate("ercreatedate");
```

DirectoryObject.getPropertyAsString()

The method returns the value of the property specified by the given name as a string.

Availability
IBM Security Identity Manager 7.0.

Synopsis
directoryObject.getPropertyAsString(name)

Arguments

name String representing the name of the property to return.

Returns
A `String` object. If the specified property does not exist, empty is returned. If the specified property has multiple values, only the first value is returned.

Description
This method returns the value of the property specified by the given name as a `String` object. If the specified property does not exist, empty string is returned. If the specified property has multiple values, only the first value is returned.

Usage

```
var name = directotyObject.getPropertyAsString("erservicename");
```

DirectoryObject.getPropertyNames()

The method returns a list of properties, such as attributes and relationships.

Availability
IBM Security Identity Manager 7.0.

Synopsis
directoryObject.getPropertyNames()

Returns
An array of Strings.

Description
This method returns a list of properties as an array of Strings. A property can be either an attribute or a relationship.

Usage `properties = directoryObject.getPropertyNames();`

DirectoryObject.name

The field represents the logical name of the object.

Availability
IBM Security Identity Manager 7.0.

Synopsis
directoryObject.name

Description
This read-only field is a string that provides the logical name of the object, represented as a String. The physical attribute used as the name can be different for each type of object.

Usage `x = directoryObject.name;`

DirectoryObject.profileName

The field returns the object profile name.

Availability
IBM Security Identity Manager 7.0.

Synopsis
directoryObject.profileName()

Description
This read-only field is a string that provides the profile name of the object, represented as a String.

Usage
```
x = directoryObject.profileName;
```

DirectoryObject.removeProperty(name)

The method removes the property specified by the given name.

Availability
IBM Security Identity Manager 7.0.

Synopsis
directoryObject.removeProperty(*name*)

Arguments

name String representing the name of the property to remove.

Description
This method removes the specified property. This change is made locally to the script environment, not to the data store. The property name can be either an attribute name or a relationship name.

Usage *directoryObject*.removeProperty("eruid");

DirectoryObject.removeProperty(name,value)

The method removes the value from the specified property.

Availability
IBM Security Identity Manager 7.0

Synopsis
directoryObject.removeProperty(*name*,*value*)

Arguments

name String representing the name of the property to be modified.

value The value to remove from the property.

Description
This method removes the specified value from property if it exists. This change is made locally to the script environment, not to the data store. The value can be a single value object or an array of objects. For multivalued objects, removeProperty(name,value) removes the values from the specified property in the directory object. The object type of the value (syntax) must be compatible with the syntax of the specified property. This method is available for the following data types:

- void removeProperty(String name, Collection value);
- void removeProperty(String name, Date value);
- void removeProperty(String name, Map value);
- void removeProperty(String name, boolean value);

- void removeProperty(String name, byte value);
- void removeProperty(String name, String value);
- void removeProperty(String name, Number value);

Usage

```
var directoryObject = Entity.get();
directoryObject.removeProperty("eraliases", "jdoe");
Entity.set(directoryObject);
```

DirectoryObject.setProperty()

The method sets the value of the specified property.

Availability

IBM Security Identity Manager 7.0.

Synopsis

directoryObject.setProperty(*name*, *value*)

Arguments

name String representing the name of the property to be created or modified.

value The value to set the property to.

Description

This method changes the value of the specified property, or adds the specified property if it does not exist. This change is made locally to the script environment, not to the data store. The value can be a single value object or an array of objects. The value type (syntax) of object must be compatible with the syntax of the specified property. This method is available for the following data types:

- `void setProperty(String name, Collection value);`
- `void setProperty(String name, Date value);`
- `void setProperty(String name, Map value);`
- `void setProperty(String name, boolean value);`
- `void setProperty(String name, byte value);`
- `void setProperty(String name, String value);`
- `void setProperty(String name, number value);`
- `void setProperty(String name, char value);`

Usage *directoryObject*.setProperty("eruid", "jdoe");

The getProperty method returns a Java array of objects that is stored in a JavaScript JavaArray object. Unlike a standard JavaScript array, JavaArray objects are used to access members of a Java array. Since Java arrays cannot be resized, the size of a JavaArray object cannot be changed. Also, JavaArray objects are typed. Setting a JavaArray element to the wrong type throws a JavaScript error.

In IBM Security Identity Manager, a JavaArray object cannot be passed directly back into a setProperty method. The JavaArray array into a standard JavaScript array as follows:

```
jsAliases = new Array();
myPerson = person.get();
aliases = myPerson.getProperty("eraliases");
for (i=0; i < aliases.length; i++) {
    jsAliases[i] = aliases[i];
}
```

```
jsAliases[aliases.length] = "myNewAlias";

myPerson.setProperty("eraliases", jsAliases);
person.set(myPerson);
```

EmailContext

The object provides access to contextual information specific to a type of notification that is sent.

Some methods for accessing information change are based upon the listed notification types. (The Reminder/Approval/RFI/WorkOrder/ComplianceAlert Notification does not support this.)

- Activity Timeout Template
- Change Account Template
- Compliance Template
- New Account Template
- New Password Template
- Process Completion Template
- Process Timeout Template
- Restore Account Template
- Suspend Account Template

Availability

IBM Security Identity Manager 7.0

Provided by

`com.ibm.itim.workflow.script.EmailContextExtension`

Synopsis

Call methods documented in this section as an `EmailContext` object. For example:

```
notificationActivity=EmailContext.getActivity();
owner=EmailContext.getAccountOwnerName()
```

Common methods

These methods are available for all types of notifications:

getActivity()

Returns information about the most recent running activity. (Returns the `ActivityInfoOC` Java Object. To get the activity information in JavaScript object, use the object, 'activity'.

getActivity(java.lang.String actDefID)

Returns information about the activity with the specified definition ID. (Returns the `ActivityInfoOC` Java Object.) This obtains information by using the `Process.$dataName.get()workflow` process. To get the activity information in JavaScript object, use '`process.getActivity(java.lang.String actDefID)`'.

getParentProcess()

Returns information about the parent process of the currently running process. (Returns the `ProcessInfoOC` Java object.) To get the process information of the parent process in JavaScript object, use '`process.getParent()`'.

getProcess()
Returns the information about the currently running process. (Returns the `ProcessInfoOC` Java object.) To get the process information of the parent process in JavaScript object, use the object, 'process'.

getRootProcess()
Returns information about the root process of the current running process. (Returns the `ProcessInfoOC` Java object.) To get the process information of the parent process in JavaScript object, use `'process.getRootProcess ()')`.

Account notification methods
These methods are available for all types of account notifications:

getAccountOwnerName()
Returns the account owner name for the account.

getAccountServiceName()
Returns the account service name for the account.

getAccountServiceProfileName()
Returns the account service profile name for the account.

getAccountUserId()
Returns the account user ID for the account.

hasNewAccess()
Returns true if the account has new access and false otherwise.

hasRemovedAccess()
Returns true if the account removed access and false otherwise.

getAccountNewAccessAsString()
Returns String that contains list of new access separated by commas.

getAccountNewAccessList()
Returns Array of String that contains the new access.

getAccountRemovedAccessAsString()
Returns a string that contains the list of removed access separated by commas.

getAccountRemovedAccessList()
Returns Array of String that contains the list of removed access.

Account Suspend/Deprovisioning Notification Methods:
These methods are only available for all types of account suspend/deprovision notifications:

getAction()
Returns the action taken against the service (resource) itself.

getReason()
Returns a descriptive reason for the deprovision.

Account New/Modify/Restore Notification Methods:
These methods are only available for all types of notifications for new, modified, and restored accounts:

showPassword()
Returns whether to display the password when the user is notified of their new account.

getAccountPassword()
Returns the account password for the account. .

getPasswordExpirePeriod()
Returns the password delivery expiration period.

getPasswordRetrievalUrl()
Returns the password delivery URL in order to retrieve the password with the accounts shared secret.

getTransactionId()
Returns the password delivery transaction ID for picking up the password created for this account.

Account Password Change Notification Methods:
These methods are available for all types of account password change notifications:

getAccountPassword()
Returns the account password for the account.

getPasswordExpirePeriod()
Returns the password delivery expiration period.

getPasswordRetrievalUrl()
Returns the password delivery URL in order to retrieve the password with the accounts shared secret.

getTransactionId()
Returns the password delivery transaction ID for picking up the password created for this account.

Enrole

The object contains the general methods.

Availability

- All JavaScript contexts
- IBM Security Identity Manager 7.0

Provided by
`com.ibm.itim.script.extensions.EnroleExtension`

Methods

generatePassword()
Generates a password for a specific service.

getAttributeValue()
Get a single value attribute value.

getAttributeValues()
Get a multi-valued attribute value.

localize()
Localized message specified in `<Message>` XML format.

log() Logs a message to the IBM Security Identity Manager log at `ERROR` level.

logError()
Logs the specified text to the IBM Security Identity Manager message log (`msg.log`) at `ERROR` level.

logInfo()
Logs the specified text to the IBM Security Identity Manager message log (`msg.log`) at `INFO` level.

logWarning()
Logs the specified text to the IBM Security Identity Manager message log (`msg.log`) at `WARN` level.

toGeneralizedTime()
Converts a time or date to generalized time format.

toMilleseconds()
Converts a String in generalized time format to an integer value in milliseconds.

traceMax()
Logs the specified text to the IBM Security Identity Manager trace log (`trace.log`) at `DEBUG_MAX` level.

traceMid()
Logs the specified text to the IBM Security Identity Manager trace log (`trace.log`) at `DEBUG_MID` level.

traceMin()
Logs the specified text to the IBM Security Identity Manager trace log (`trace.log`) at `DEBUG_MIN` level.

Description
Provides some common utilities for use in many different scripting contexts.

Enrole.generatePassword()

The method generates a new valid password for an account.

Availability
`generatePassword()` requires a service to work, so `generatePassword()` is only available when the `ServiceExtension` is used.

Synopsis
```
Enrole.generatePassword()
```

Returns
A String that is a valid password for the Service `DirectoryObject` stored in the "service" variable.

Description
This method generates a new valid password for a service.

Enrole.getAttributeValue()

The method retrieves the attribute's value.

Availability
Deprecated as of IBM Security Identity Manager 7.0. Replace with `DirectoryObject.getProperty()`

Synopsis
```
Enrole.getAttributeValue(name, defaultValue)
```

Arguments

name String representing the name of the property to return.

defaultValue
Default value to return if there is no value to return.

Returns
An Object. The type of object returned depends on the property obtained. If the specified property does not exist, the default value is returned.

Description
This method retrieves the value of the specified property.

Enrole.getAttributeValues()

The method retrieves a multi-valued attribute value.

Availability
Deprecated as of IBM Security Identity Manager 7.0. Replace with `DirectoryObject.getProperty()`

Synopsis
`Enrole.getAttributeValues(`*name*`)`

Arguments

name String representing the name of the property to return.

Returns
An array of objects. The type of object returned depends on the property obtained. If the specified property does not exist, an empty array is returned.

Description
This method retrieves the value of the specified property.

Enrole.localize()

The method localizes a message specified in <Message> XML format.

Availability
IBM Security Identity Manager 7.0

Synopsis
`Enrole.localize(String xmlMsg, String localStr)`

Arguments

xmlMsg
A message specified in XML.

localStr
A String that represents the locale to be used for globalization.

Returns
AA localized message.

Description
This method globalizes an XML message to the specified locale.

Enrole.log()

The method logs messages to the IBM Security Identity Manager message log (`msg.log`).

Availability
IBM Security Identity Manager 7.0

Synopsis
`Enrole.log(`*`category`*`, `*`message`*`);`

Arguments

category
The category of the log entry, entered as a String. The category argument can be used or it can be left empty, but the argument must not be null.

message
The message to be logged, entered as a String.

Description
Logs a message to the IBM Security Identity Manager log at error level.

Usage

```
var roleDN = ..;(DN of role)
var role = new Role(roleDN);

//  Put next statement on one line

Enrole.log("script", "The role name is
 "+ role.getProperty("errolename")[0]);
```

Use the following new methods in IBM Security Identity Manager 7.0 to provide greater adaptability, control, or flexibility over the `Enrole.log()` method:

- logError()
- logInfo()
- logWarning()
- traceMax()
- traceMid()
- traceMin()

Enrole.logError()

The method logs text messages to the IBM Security Identity Manager message log (`msg.log`) with a message severity level of `ERROR`.

Availability
IBM Security Identity Manager 7.0

Synopsis
`Enrole.logError(`*`component`*`, `*`method`*`, `*`message`*`);`

Arguments

component
The component of the log entry, entered as a String. The component can be any string. Logging can be controlled for components by setting specific log levels in the `enRoleLogging.properties` file.

method
The string to display in the "Method" record of the message log. Useful to point where in the script the message originated.

message
The string to represent the contents of the message log to be written to the log file.

Description
Writes an error message to the IBM Security Identity Manager message log (`msg.log`).

Usage An example to write a `msg.log` message at `ERROR` level with the component name `com.ibm.myExtension` and the method name `postScriptOfAccountCreate`:

```
var userName = "Joe";
 // below is a single line
 Enrole.logError("com.ibm.myExtension","postScriptOfAccountCreate",
"Recording error message after unsuccessful account creation for user "
    + userName + ".");
```

Enrole.logInfo()

The method logs text messages to the IBM Security Identity Manager message log (`msg.log`) with a message severity level of `INFO`.

Availability
IBM Security Identity Manager 7.0

Synopsis
`Enrole.logInfo(`*`component`*`, `*`method`*`, `*`message`*`);`

Arguments

component
The component of the log entry, entered as a String. The component can be any string. Logging can be controlled for components by setting specific log levels in the `enRoleLogging.properties` file.

method
The string to display in the "Method" record of the message log. Useful to point where in the script the message originated.

message
The string to represent the contents of the message log to be written to the log file.

Description
Writes an error message to the IBM Security Identity Manager message log (`msg.log`).

Usage An example to write a `msg.log` message at `INFO` level with the component name `com.ibm.myExtension` and the method name `postScriptOfAccountCreate`:

```
var userName = "Joe";
 // below is a single line
 Enrole.logInfo("com.ibm.myExtension","postScriptOfAccountCreate",
"Recording information message after account creation for user " + userName + ".");
```

Enrole.logWarning()

The method logs text messages to the IBM Security Identity Manager message log (`msg.log`) with a message severity level of `WARN`.

Availability
IBM Security Identity Manager 7.0

Synopsis

`Enrole.logWarning((component, method, message);`

Arguments

component
The component of the log entry, entered as a String. The component can be any string. Logging can be controlled for components by setting specific log levels in the `enRoleLogging.properties` file.

method
The string to display in the "Method" record of the message log. Useful to point where in the script the message originated.

message
The string to represent the contents of the message log to be written to the log file.

Description

Writes a warning message to the IBM Security Identity Manager message log (`msg.log`).

Usage An example to write a `msg.log` message at `WARN` level with the component name `com.ibm.myExtension` and the method name `postScriptOfAccountCreate`:

```
var userName = "Joe";
 // below is a single line
 Enrole.logWarning("com.ibm.myExtension","postScriptOfAccountCreate",
"Recording warning message after account creation for user " + userName + ".");
```

Enrole.toGeneralizedTime()

The method converts a time or date to generalized time format.

Availability

IBM Security Identity Manager 7.0.

Synopsis

`Enrole.toGeneralizedTime(time)`

Arguments

time Integer time in milliseconds or a Date object.

Description

This method converts a time or date to generalized time format. Can be used in either Identity Policies or in default entitlements.

Usage

```
genTime = Enrole.toGeneralizedTime(seconds);
```

Enrole.toMilliseconds()

The method converts a string in generalized time format to an integer value in milliseconds.

Availability

IBM Security Identity Manager 7.0.

Synopsis

`Enrole.toMilliseconds(genTime)`

Arguments

genTime
String in generalized time format.

Description
This method converts a String in generalized time format to an integer value in milliseconds.

Usage `seconds = Enrole.toMilliseconds(genTime);`

Enrole.traceMax()

The method logs text messages to the IBM Security Identity Manager trace log (`trace.log`) with a message severity level of `DEBUG_MAX`.

Availability
IBM Security Identity Manager 7.0

Synopsis
`Enrole.traceMax(`*`(component`*`, `*`method`*`, `*`message`*`);`

Arguments

component
The component of the log entry, entered as a String. The component can be any string. Logging can be controlled for components by setting specific log levels in the `enRoleLogging.properties` file.

method
The string to display in the "Method" record of the message log. Useful to point where in the script the message originated.

message
The string to represent the contents of the trace message to be written to the log file.

Description
Writes a `DEBUG_MAX` message to the IBM Security Identity Manager trace log (`trace.log`).

Usage An example to write a `trace.log` message at `DEBUG_MAX` level with the component name `com.ibm.myExtension` and the method name `postScriptOfAccountCreate`:

```
var userName = "Joe";
 // below is a single line
 Enrole.traceMax("com.ibm.myExtension","postScriptOfAccountCreate",
"Recording DEBUG_MAX trace message after account creation for user " + userName + ".");
```

Enrole.traceMid()

Logs text messages to the IBM Security Identity Manager trace log (`trace.log`) with a message severity level of `DEBUG_MID`.

Availability
IBM Security Identity Manager 7.0

Synopsis
`Enrole.traceMid(`*`(component`*`, `*`method`*`, `*`message`*`);`

Arguments

component
The component of the log entry, entered as a String. The component can be any string. Logging can be controlled

for components by setting specific log levels in the enRoleLogging.properties file.

method
The string to display in the "Method" record of the message log. Useful to point where in the script the message originated.

message
The string to represent the contents of the trace message to be written to the log file.

Description
Writes a DEBUG_MID message to the IBM Security Identity Manager trace log (trace.log).

Usage An example to write a trace.log message at DEBUG_MID level with the component name com.ibm.myExtension and the method name postScriptOfAccountCreate:

```
var userName = "Joe";
 // below is a single line
 Enrole.traceMid("com.ibm.myExtension","postScriptOfAccountCreate",
"Recording DEBUG_MID trace message after account creation for user " + userName + ".");
```

Enrole.traceMin()

The method logs text messages to the IBM Security Identity Manager trace log (trace.log) with a message severity level of DEBUG_MIN.

Availability
IBM Security Identity Manager 7.0

Synopsis
Enrole.traceMin(*(component*, *method*, *message*);

Arguments

component
The component of the log entry, entered as a String. The component can be any string. Logging can be controlled for components by setting specific log levels in the enRoleLogging.properties file.

method
The string to display in the "Method" record of the message log. Useful to point where in the script the message originated.

message
The string to represent the contents of the trace message to be written to the log file.

Description
Writes a DEBUG_MIN message to the IBM Security Identity Manager trace log (trace.log).

Usage An example to write a trace.log message at DEBUG_MIN level with the component name com.ibm.myExtension and the method name postScriptOfAccountCreate:

```
var userName = "Joe";
 // below is a single line
 Enrole.traceMin("com.ibm.myExtension","postScriptOfAccountCreate",
"Recording DEBUG_MIN trace message after account creation for user " + userName + ".");
```

Error

This object contains a script error description to notify the calling code of an exceptional runtime condition.

When an error is returned from a script evaluation, it is converted to a Java exception and thrown from the script evaluator class.

Availability
IBM Security Identity Manager 7.0

Provided by
`com.ibm.itim.script.extensions.EnroleExtension`

Methods

setMessage()
Sets the message for the error.

getMessage()
Retrieves the error message for the error.

setErrorCode()
Sets the error code for the error.

getErrorCode()
Retrieves the error code for the error.

Usage

```
var sn = subject.getProperty("sn");
if(sn == null || sn.length == 0) {
   error.setMessage("sn was missing");
   return error;
} else {
  return sn[0];
}
```

Error.setMessage()

The method sets the message for the error.

Availability
IBM Security Identity Manager 7.0

Synopsis
`error.setMessage(`*`String msg`*`)`

Arguments

msg String representing the message to be set.

Description
This method sets the text for an error message. The function takes in one String parameter.

Usage `error.setMessage("sn was missing");`

Error.getMessage()

The method retrieves the message set for an error.

Availability
IBM Security Identity Manager 7.0.

Synopsis
`error.getMessage()`

Returns
String message for an error.

Description
This method retrieves the text of an error message.

Usage `messageValue = error.getMessage();`

Error.setErrorCode()

The method sets the error code for the error.

Availability
IBM Security Identity Manager 7.0.

Synopsis
`error.setErrorCode(int code)`

Arguments

code Integer representing the error code.

Description
This method sets the error code for an error message. The function takes in one **Int** parameter.

Usage `error.setErrorCode(1);`

Error.getErrorCode()

The method retrieves the error code set for an error.

Availability
IBM Security Identity Manager 7.0.

Synopsis
`error.getErrorCode()`

Returns
Integer value for an error code.

Description
This method retrieves the error code of an error message.

Usage `errorCodeValue = error.getErrorCode();`

ExtendedPerson

This object extends the Person object with the ownership type information for account adoption.

Availability
IBM Security Identity Manager 7.0.

Inherited from
Person.

Provided by
`com.ibm.itim.script.extensions.model.PersonModelExtension`

Ownership type
INDIVIDUAL

String constant represents the default ownership type.

Constructor

```
new ExtendedPerson(dn)
```

Arguments

DN DN string of a specific person entity.

Returns

The new `ExtendedPerson` object that represents a person with the DN and INDIVIDUAL ownership type.

```
new ExtendedPerson(dn, ownershipType)
```

Arguments

DN DN string of a specific person entity.

ownershipType

String representing one of the ownership types configured in IBM Security Identity Manager.

Returns

The new `ExtendedPerson` object that represents a person with the DN and ownership type. If the ownership type is invalid, it throws `ScriptException`.

```
new ExtendedPerson(person)
```

Arguments

person

Person object.

Returns

The new `ExtendedPerson` object that represents the person with the INDIVIDUAL ownership type.

```
new ExtendedPerson(person, ownershipType)
```

Arguments

person

Person object.

ownershipType

String representing one of the ownership types configured in IBM Security Identity Manager.

Returns

The new `ExtendedPerson` object that represents the person with the ownership type. If the ownership type is invalid, it throws `ScriptException`.

Methods

getOwnershipType()

Returns the ownership type.

setOwnershipType()

Sets the ownership type.

ExtendedPerson.getOwnershipType()

The method return the ownership type as a string.

Availability

IBM Security Identity Manager 7.0.

Synopsis

```
ExtendedPerson.getOwnershipType()
```

Returns

String.

Description

This method returns the ownership type.

Usage

```
var ownershipType = extendedPerson.getOwnershipType();
```

ExtendedPerson.setOwnershipType()

The method sets the value of the ownership type.

Availability

IBM Security Identity Manager 7.0.

Synopsis

```
ExtendedPerson.setOwnershipType(value)
```

Arguments

value A string represents one of the ownership types configured in IBM Security Identity Manager.

Description

This method updates the ownership type. If the ownership type is invalid, it throws `ScriptException`.

Usage

```
var extendedPerson.setOwnershipType("System");
```

IdentityPolicy

The object represents the identity policy entity.

Availability

IBM Security Identity Manager 7.0
Identity Policy context

Provided by

`com.ibm.itim.policy.script.IdentityPolicyExtension`

Methods

getNextCount()
Returns a number that can be appended to the end of a user name to make that user name unique.

userIDExists()
Checks if requested `UID` is already in use.

Description

This object represents a IBM Security Identity Manager Policy entity.

IdentityPolicy.getNextCount()

The method gets a number that can be appended to the end of a user name to make that user name unique. `ServiceExtension` must be loaded for `getNextCount()` to work.

Availability

IBM Security Identity Manager 7.0

Synopsis

```
IdentityPolicy.getNextCount(baseId)
```

Arguments

baseId
The base user name.

Returns
A number that can be appended to the end of a user name to make the user name unique. Returns `-1` if the user name is already unique and `-2` if an error occurs.

Description
This method checks whether requested `UID` is already in use.

Usage

```
num = IdentityPolicy.getNextCount(baseId);
return baseId + num;
```

IdentityPolicy.userIDExists()

The method checks if the requested `UID` is in use.

Availability
IBM Security Identity Manager 7.0.

Synopsis

`IdentityPolicy.userIDExists(`*`uid`*`, `*`checkAllServices`*`, `*`checkRecycleBin`*`)`

Arguments

uid User identity.

checkAllServices
If set to true, all service instances are checked to see whether the uid is used on an account of any service type. If set to false, only the target service instance is checked. This argument is optional. Default value is false.

checkRecycleBin
If set to true, the recycle bin is checked for any deleted accounts. This parameter is intended to work in conjunction with the **`checkAllServices`** parameter. Set this parameter true only when the **`checkAllServices`** parameter is also set to true. This argument is optional. Default value is false.

Returns
True if the user ID exists, false otherwise.

Description
This method checks whether the requested `UID` is in use.

Usage

```
// To create a user ID without checking for it in the recycle bin but
// checking it against all services.
tf = IdentityPolicy.userIDExists("jason_jones", true, false);
```

PackagedApprovalDocument

A relevant data object used in multi-item approval, used exclusively in user recertification workflows. This object is made up of multiple `PackagedApprovalItem` objects from the user recertification approval and allows for searching and retrieving recertification items.

Availability

IBM Security Identity Manager 7.0.

Constructor

`new PackagedApprovalDocument()`

Constructs an empty approval document object. Instances might also be obtained in user recertification workflow and notifications by accessing the relevant data item `"ApprovalDocument."` For example, `ApprovalDocument.get()` returns a `PackagedApprovalDocument` in a user recertification workflow.

Properties

TYPE_ACCOUNT

A constant for approval items that are accounts.

TYPE_GROUP

A constant for approval items that are groups on other services but are not defined as an access.

TYPE_GROUP_ACCESS

A constant for approval items that are groups and also defined as accesses.

TYPE_ITIM_GROUP

A constant for approval items that are groups on services of type ITIM Service.

TYPE_ROLE

A constant for approval items that are roles.

Methods

addItem(PackagedApprovalItem item)

Returns a Boolean flag that indicates that a `PackagedApprovalItem` item is added in this approval document.

containsDecisionCode(decisionCode)

Returns a Boolean flag that indicates whether any of the items in this document that allow for decisions contain the specified decision code string. Valid decision codes are `activity.APPROVED` and `activity.REJECTED`.

getDecisionItemCountByType(type)

Returns the number of items in this document that support decisions and have the specified type. The types are defined as constants on this object, such as `TYPE_ROLE` or `TYPE_ACCOUNT`. This method considers all approval items in the document that supports decisions, including children of top-level items.

getDecisionItemCountByType(type, includeChildren)

Returns the number of items in this document that support decisions and have the specified type. The types are defined as constants on this object, such as `TYPE_ROLE` or `TYPE_ACCOUNT`. Depending on the value the `includeChildren` flag, this method

might also count all items in this document, including any items that are children of the top-level items.

getItemCountByType(type)
Returns the number of items in this document that are of the specified type. The types are defined as constants on this object, such as `TYPE_ROLE` or `TYPE_ACCOUNT`. This method considers all approval items in the document, including children of top-level items.

getItemCountByType(type, includeChildren)
Returns the number of items in this document that are of the specified type. The types are defined as constants on this object, such as `TYPE_ROLE` or `TYPE_ACCOUNT`. Depending on the value of the `includeChildren` flag, this method might also count all items in this document, including any items that are children of the top-level items.

getItemCountByTypeAndDecision(type, decisionCode)
Returns the number of items in this document that are of the specified type and that allow for decisions and contain the specified decision code string. The types are defined as constants on this object, such as `TYPE_ROLE` or `TYPE_ACCOUNT`. Valid decision codes are `activity.APPROVED` and `activity.REJECTED`. This method considers only top-level approval items and does not count the children of those items.

getItemsByType(type)
Returns the top-level items in this approval document that have the specified type as an array of `PackagedApprovalItem` objects. The types are defined as constants on this object, such as `TYPE_ROLE` or `TYPE_ACCOUNT`.

getItemsByTypeAndDecision(type, decisionCode)
Returns the top-level items in this approval document that have the specified type. If decisions are allowed, it contains the specified decision code string as an array of `PackagedApprovalItem` objects. The types are defined as constants on this object, such as `TYPE_ROLE` or `TYPE_ACCOUNT`. Valid decision codes are `activity.APPROVED` and `activity.REJECTED`.

removeItem(String identifier)
Returns a Boolean flag that indicates that a `PackagedApprovalItem` that corresponds to the `identifier` is removed from this approval document.

setDecisionCodeForAllItems(decisionCode)
Sets the specified `decisionCode` on all items in this document, including any children of top-level items. Any items that do not support decisions are skipped. Valid decision codes are `activity.APPROVED` and `activity.REJECTED`.

Description
The object represents the multi-item approval document in the JavaScript environment.

PackagedApprovalItem

A relevant data object used in IBM Security Identity Manager multi-item approval, used exclusively in user recertification workflows. This object represents the individual roles, accounts, and groups that are presented to the user during the recertification process. Some items might contain a decision code that indicates the choice of the approvers for that item. Each item also contains a list of children that is used to represent relationships between accounts and groups.

Availability

IBM Security Identity Manager 7.0.

Constructor

`new PackagedApprovalItem(itemType, value)`

Constructs a `PackagedApprovalItem` object that does not support decisions and is read-only during the recertification approval activity. The parameters are an item type constant and value, where the value is a `DirectoryObject` that matches the type, such as Role or Account.

`new PackagedApprovalItem(itemType, value, decisionCode)`

Constructs a `PackagedApprovalItem` object that supports decisions. The **`decisionCode`** parameter is either `activity.APPROVED`, `activity.REJECTED`, or `null`, where null indicates that a decision is required but not yet specified.

For example:

```
new PackagedApprovalItem(PackagedApprovalDocument.TYPE_ACCOUNT, acctObj)

new PackagedApprovalItem(PackagedApprovalDocument.TYPE_ROLE, roleObj, activity.APPROVED)
```

Properties

DECISION_NOT_APPLICABLE
A constant for approval items that do not support decisions and are read-only during the recertification.

Methods

getItemTypeString()
Returns the type of the item, where the constant values are defined on the `PackagedApprovalDocument` object (`TYPE_ROLE`, `TYPE_ACCOUNT`, `TYPE_GROUP`, `TYPE_GROUP_ACCESS`).

getDecisionCode()
Returns the decision code for this item, where the possible values are `activity.APPROVED` and `activity.REJECTED`. This method might also return `PackagedApprovalItem.DECISION_NOT_APPLICABLE` if this item is for informational purposes only, or `null` if the decision is not yet specified.

getValue()
Returns a `DirectoryObject` for the role, account, or group of this item.

getChildItems()
Returns an array of `PackagedApprovalItem` objects that are the children of this item. For example, account items can have groups as their children.

getChildItemsByDecision(decisionCode))
Returns an array of `PackagedApprovalItem` objects that are the

children of this item and have the specified decision code, such as `activity.APPROVED` or `activity.REJECTED`.

Description

The Object represents the Security Identity Manager multi-item approval element in the JavaScript environment.

Participant

Workflow participant entity, which specifies an activity participant. In a mail node, this entity specifies the mail recipient.

Participant applies only to manual activity types, including `Approval`, `RFI`, `WorkOrder`, and `Mail`.

The participant of an activity can be specified during workflow design as Custom Defined Participant. In this case, the Participant JavaScript object can be used to construct the appropriate participant based on the process context.

Availability

IBM Security Identity Manager 7.0

Provided by

`com.ibm.itim.workflow.script.WorkflowExtension`

Constructor

`new Participant(type, dn)`

Arguments

type Code that categorizes the participant type.

dn Optional DN of a specific entity.

Returns

The newly created and initialized participant object.

Constructor for custom self approval

`new Participant(type, boolean)`

Arguments

type Type is either `REQUESTEE` or `REQUESTOR`.

boolean

Self approval values. `true` enables the custom self approval workflow. `false` disables the custom self approval workflow.

Returns

The newly created and initialized participant object.

Properties

implementation

This property contains JavaScript that returns participant when the participant type is Custom.

name Identifies the participant.

type Code that categorizes the participant type.

Description

The participant specifies an activity participant. Participant applies only to

manual activity types, including `Approval`, `RFI`, `Work Order` and `Mail` activities. The participant of an activity or recipient of a mail activity can be specified during workflow design as Custom Defined Participant. In this case, the Participant JavaScript object can be used to construct the appropriate participant based on the process context.

Usage

```
//assume person is one of the relevant data in the workflow
//process for the target user involved
if( person.get().getProperty("title")[0]=="Manager" )
  return new Participant(ParticipantType.SYSTEM_ADMIN);
else
  return new Participant(ParticipantType.SUPERVISOR);


//assume person is one of the relevant data in the workflow
//process for the target user involved
if( person.get().getProperty("title")[0]=="Manager")
  return new Participant(ParticipantType.USER, person.get().dn);
else
  ...
```

Participant.implementation

The field represents the custom defined participant.

Availability
IBM Security Identity Manager 7.0.

Synopsis
participant.implementation

Description
This read-only field is a string that provides the custom-defined participant, which contains the JavaScript code to return the participant.

Usage `x = participant.implementation;`

Participant.name

The field represents the DN of the participant.

Availability
IBM Security Identity Manager 7.0.

Synopsis
participant.name

Description
This read-only field is a Distinguished Name that identifies the participant. It is only applicable to participant types of `ROLE` and `USER`.

Usage `x = participant.name;`

Participant.type

the field represents the code that categorizes the participant type.

Availability
IBM Security Identity Manager 7.0.

Synopsis
participant.type

Description
This read-only field is a string that represents a code that categorizes the participant type.

Usage `x = participant.type;`

ParticipantType

An entity that represents the workflow participant type constants.

Availability
IBM Security Identity Manager 7.0.

Provided by
`com.ibm.itim.workflow.script.WorkflowExtension`

Properties

DOMAIN_ADMIN
Participant type for the domain administrator of the organizational container. It is associated with the Subject account service (as specified by the Subject context in the workflow properties window).

```
participant = new Participant(ParticipantType.DOMAIN_ADMIN);
```

REQUESTOR

Enables the self approval by requester for specific workflow even though the global configuration is set to disable the self approval. By setting the value of boolean to `true`, self approval for specific workflow is enabled.

```
participant = new Participant(ParticipantType.REQUESTOR, boolean);
```

Participant type for the person that initiated the request. If a person initiates a change request for a person that triggers policy enforcement, the participant is the person that requests the change. For data loads, the participant is the system user. Use the Update Property page from the **Appliance Dashboard** of the IBM Security Identity Manager virtual appliance console to set the following property in `enRole.properties` to `true`. See Managing the server properties. An approval request that has the requester as the participant is automatically approved by the system:

```
participant = new Participant(ParticipantType.REQUESTOR);
```

REQUESTEE
Participant type for the person designated as the requestee in the owner field of the relevant data.

```
participant = new Participant(ParticipantType.REQUESTEE);
```

Enables the self approval by requestee for specific workflow even though the global configuration is set to disable the self approval. By setting the value of boolean to `true`, self approval for specific workflow is enabled.

```
participant = new Participant(ParticipantType.REQUESTEE, boolean);
```

ROLE Participant type for a specific organizational role. All user members of the role and its child roles are notified and are eligible to respond, the first response triggers the workflow to continue. In other words, specifying a role cannot be used to require multiple participants to approve the request.

```
participant = new Participant(ParticipantType.ROLE, roleDN);
```

ROLE_OWNER
Participant type for the owner of the role (if specified). The Role is resolved based on the owners specified in the `OrgRole` listed as an input parameter for the operational workflow operation. If there is no `OrgRole` specified as an input parameter in the workflow, the participant is not resolved.

```
participant = new Participant(ParticipantType.ROLE_OWNER);
```

SERVICE_OWNER
Participant type for the owner of the service (if specified). The Service is resolved based on the account object from the workflow relevant data that is marked as "Subject" in the properties window.

```
participant = new Participant(ParticipantType.SERVICE_OWNER);
```

SOD_POLICY_OWNER
Participant type for the owners of the separation of duty policy (if specified). The owners are resolved based on the `SeparationOfDutyRuleViolation` object from the workflow relevant data that is marked as "Subject" in the properties window. If there is no `SeparationOfDutyRuleViolation` specified as the Subject of the workflow, the participant is not resolved.

The SOD_POLICY_OWNER participant type is used only in the `approveSoDViolation` global operation.

```
participant = new Participant(ParticipantType.SOD_POLICY_OWNER);
```

SPONSOR
Participant type for the person designated as the sponsor with the sponsor relationship for the requestee (as marked in relevant data).

```
participant = new Participant(ParticipantType.SPONSOR);
```

SUPERVISOR
Participant type for the supervisor or manager of the requestee. If none is specified for the requestee, then the supervisor designated on the organizational container of the requestee becomes the participant. If no supervisor is specified for the organizational container of the requestee, then the next level up is checked for a supervisor. The search continues up the tree until the top of the organization is reached. If no supervisor is found, the participant is unresolved.

```
participant = new Participant(ParticipantType.SUPERVISOR);
```

SYSTEM_ADMIN
Participant type for a member of the Security Identity Manager System Administrator group.

```
participant = new Participant(ParticipantType.SYSTEM_ADMIN);
```

USER Participant type for a specific person to respond to the request. The person must have a Security Identity Manager account.

```
participant = new Participant(ParticipantType.USER, userDN);
```

ITIM GROUP
Participant type for a specific `ITIM` group. Though all members of the group are notified, and all are eligible to respond, the first response triggers the workflow to continue. Specifying a group cannot be used to require multiple participants to approve the request.

```
participant = new Participant(ParticipantType.GROUP, groupDN);
```

Description
This entity represents the workflow participant type constants.

Person

The object represents the person entity.

Availability
IBM Security Identity Manager 7.0.

Provided by
`com.ibm.itim.script.extensions.model.PersonModelExtension`

Inherits From
`DirectoryObject`

Constructors

new Person(String dn)
Arguments:

dn The distinguished name of a specific person entry in the directory server.

Returns: A new Person object that represents the person with the given DN.

new Person(DirectoryObject directoryObject)
Arguments:

directoryObject
`DirectoryObject` to be contained in the person

new Person(DirectoryObjectEntity directoryObjectEntity)
Arguments:

directoryObjectEntity
`DirectoryObjectEntity` to be contained in the person

Methods

getAllAssignmentAttributes()
Returns an array of the `RoleAssignmentAttribute` objects that are defined in all of authorized roles for this person. The authorized roles consist of both the direct roles for this person and also all of the parent roles of the direct roles.

getAndDecryptSynchPassword()
Decrypts and returns the decrypted synch password of the person entity in plain text.

Note: This method is available in the scripting context of IBM Security Identity Manager only if the **`javascript.password.access.enabled`** property is set to `true` in the *ISIM_HOME*`/data/scriptframework.properties` file.

getAndDecryptPersonPassword()
Decrypts and returns the decrypted person password of the person entity in plain text.

Note: This method is available in the scripting context of Security Identity Manager only if the **javascript.password.access.enabled** property is set to `true` in the *ISIM_HOME*`/data/scriptframework.properties` file.

getRoleAssignmentData()
Returns all role assignment data for the person.

getRoleAssignmentData(String roleAssignedDN)
Returns all role assignment data for the person for the specified role.

getRoles()
Returns an array of DirectoryObjects, each representing a role.

getNewRoles()
Returns an array of newly added roles for the person.

getNewRoles(boolean convertReplaceToAddDelete)
Returns an array of newly added static roles for the person.

getRemovedRoles()
Returns an array of removed roles for the person.

getRemovedRoles(boolean convertReplaceToAddDelete)
Returns an array of removed static roles for the person.

isInRole(String roleName)
Determines whether the person belongs to the role. Returns Boolean.

removeRole()
Removes the person from the specified role.

removeRoleAssignmentData(String roleAssignedDN)
Removes all role assignment data for the person from the specified role.

updateRoleAssignmentData(RoleAssignmentObject[] roleAssignmentObject)
Updates a person with the role assignment attribute value changes that are defined in the set of `RoleAssignmentObjects`.

Person.getAllAssignmentAttributes()

The method returns an array of the `RoleAssignmentAttribute` objects that are defined for all of authorized roles for this person. The authorized roles consist of both the direct roles for this person and also all the parent roles of the direct roles.

Availability
IBM Security Identity Manager 7.0

Synopsis
person`.getAllAssignmentAttributes()`

Arguments
None

Description
This method is defined on the Person object. It returns an array of the `RoleAssignmentAttribute` objects that are defined in all of authorized roles for this person. The authorized roles consist of both the direct roles for this person and also all the parent roles of the direct roles. The method returns

an empty array if no assignment attribute exists. `RoleAssignmentAttribute` objects contains role assignment attribute name, role name, and role DN.

Usage

```
//The script is used in a workflow, in which Entity is a person object.
var person = Entity.get();

//get assignment attributes of the person
var attributeList = person.getAllAssignmentAttributes();
if (attributeList.length == 0) {
    Enrole.log("script", "No assignment attribute for this role: "
        + role.name);
    return;
}

// print out the role assignment attribute name.
for (var i=0; i < attributeList.length; i++) {
    var roleAtr = attributeList[i];
    Enrole.log("script","attribute name-----: "+ roleAtr.getName());
}
```

Person.getAndDecryptSynchPassword()

The method decrypts and returns the decrypted sync password of the person entity in plain text.

Availability

IBM Security Identity Manager 7.0.

Synopsis

person`.getAndDecryptSynchPassword()`

Arguments

None

Description

This method is defined on the Person object. It returns a string that represents the plain text sync password for the person that is used for synchronization. It decrypts and returns the decrypted sync password set in the person object. This function returns null if the sync password is not present. This method can be used in IBM Security Identity Manager scripting context if the **`javascript.password.access.enabled`** property is set to `true` in the `scriptframework.properties` file.

Use the Update Property page to work with the `scriptframework.properties` file. See Managing the server properties.

Usage

```
//The script is used in a workflow, in which Entity is a person object.
var person = Entity.get();
//get sync password set on the person
var synchPassword = person.getAndDecryptSynchPassword();
```

Person.getAndDecryptPersonPassword()

The method decrypts and returns the decrypted password of the person entity in plain text.

Availability

IBM Security Identity Manager 7.0.

Synopsis

person`.getAndDecryptPersonPassword()`

Arguments

None

Description

This method is defined on the Person object. It returns a string that represents the plain text password for the person. It decrypts and returns the decrypted password set in the person object. This function returns null if the password is not present. This method can be used in IBM Security Identity Manager scripting context if the **javascript.password.access.enabled** property is set to `true` in the `scriptframework.properties` file.

Use the Update Property page to work with the `scriptframework.properties` file. See Managing the server properties.

Usage

```
//The script is used in a workflow, in which Entity is a person object.
var person = Entity.get();
//get person password set on the person
var personPassword = person.getAndDecryptPersonPassword();
```

Person.getRoleAssignmentData()

The method returns all the role assignment data for the person, as an array of `RoleAssignmentObject` objects that contain the role assignment values, defined Role DN and assigned Role DN.

Availability

IBM Security Identity Manager 7.0

Synopsis

person`.getRoleAssignmentData()`

Arguments

none

Description

This method is defined on the Person object. It returns an array of `RoleAssignmentObject` objects, containing the role assignment values, defined Role DN, and assigned Role DN. The method returns an empty array if no assignment data exists.

Usage

```
//The script is used in a workflow, in which Entity is a person object.
var person = Entity.get();
 var assignmentObjects = person.getRoleAssignmentData();
 if (assignmentObjects.length == 0) {
  Enrole.log("script", "There is no assignment values for " + person.name);
  return;
 }
 var str = "The number of role assignment objects returned from
      person.getRoleAssignmentData(): " +
     assignmentObjects.length + "\n";
 for(var i=0; i<assignmentObjects.length; i++) {
  var obj = assignmentObjects[i];
  str += obj.toString() + "\n";
 }
 Enrole.log("script", "The assignment attribute data for person:"+
      person.name+" is:"+ str);
```

Person.getRoleAssignmentData(String roleAssignedDN)

The method returns all the role assignment data for the person. The data is an array of `RoleAssignmentObject` objects that contain the role assignment values, defined Role DN, and assigned Role DN for the specified assigned role.

Availability

IBM Security Identity Manager 7.0

Synopsis

person`.getRoleAssignmentData(String roleAssignedDN)`

Arguments

roleAssignedDN

The distinguished name of the assigned role

Description

This method is defined on the Person object. It returns an array of `RoleAssignmentObject` objects, containing the role assignment values, defined Role DN, and assigned Role DN for a specified assigned role. The method returns an empty array if no assignment data exists.

Usage

```
//The script is used in a workflow, in which Entity is a person object.
var person = Entity.get();
var roleDNs = person.getProperty("erroles");
if(roleDNs.length == 0) {
 Enrole.log("script", person.name + " does not have any role");
 return;
}
// Get role assignment data for the first role.
var roleDN = roleDNs[0];
var role = new Role(roleDN);
var assignmentObjects = person.getRoleAssignmentData(roleDNs[0]);
if (assignmentObjects.length == 0) {
 Enrole.log("script", person.name + " does not have any assignment
   objects for role: + role.name);
 return;
}
var str = "The number of role assignment objects returned from
    person.getRoleAssignmentData() for "
    + role.name + " :" + assignmentObjects.length + "\n";
for(var i=0; i<assignmentObjects.length; i++) {
 var obj = assignmentObjects[i];
 str += obj.toString() + "\n";
}
Enrole.log("script", str);
```

Person.getRoles()

The method returns roles assigned to a Person.

Availability

IBM Security Identity Manager 7.0

Synopsis

`person.getRoles()`

Description

This method defined on the Person object returns an array of roles that the person belongs to. The return type is an array of entities, which are instances of role directory entity objects. The properties available on the Entity Objects are `name` and `description`.

Usage

```
// logs the names of all roles that a person belongs to
var per = person.get();
var rolesArray = per.getRoles();
if(rolesArray.length>0){
   Enrole.log("script", per.getProperty("cn")[0] +
      " belongs to following roles: ");

   for( var i=0; i<rolesArray.length;i++) {
      Enrole.log("script",
         rolesArray[i].getProperty("errolename")[0]);
   }
} else {
   Enrole.log("script", per.getProperty("cn")[0] +
      "does not belong to any roles");
}
```

Person.getNewRoles()

The method returns an array of newly added static roles for a Person.

Availability

IBM Security Identity Manager 7.0

Synopsis

person.getNewRoles()

Description

This method defined on the person object returns an array of new static roles associated with the person. The return type is an array of DirectoryObjects,

Note: The person object is often a runtime object in memory, and these new static roles were not added to the directory.

Usage

```
var newRoles = per.getNewRoles();
```

Person.getNewRoles(boolean convertReplaceToAddDelete)

The method returns an array of newly added static roles for a Person.

Availability

IBM Security Identity Manager 7.0

Synopsis

person.getNewRoles(boolean convertReplaceToAddDelete)

Description

This method defined on the person object returns an array of new static roles associated with the person. The return type is an array of DirectoryObjects.

Depending on how the person modification is triggered, the role changes get stored as REPLACE operation or ADD/DELETE operation.

When the method is invoked with a true parameter, the REPLACE operation is evaluated as ADD/DELETE and the method will return only the newly added roles.

When the method is invoked with a false parameter, if role changes are stored as REPLACE operation, the method will return the complete set of new roles. This will include the existing roles too.

Note: The person object is often a runtime object in memory, and these new static roles were not added to the directory.

Usage

```
var newRoles = per.getNewRoles(true);
```

Person.getRemovedRoles()

The method returns an array of removed static roles for the `Person`.

Availability

IBM Security Identity Manager 7.0

Synopsis

person`.getRemovedRoles()`

Description

This method defined on the person object returns an array of static roles from which the person was removed. The return type is an array of `DirectoryObjects`.

Note: The person object is often a runtime object in memory, and these static roles were not removed from the directory.

Usage

```
var removedRoles = per.getRemovedRoles();
```

Person.getRemovedRoles(boolean convertReplaceToAddDelete)

The method returns an array of removed static roles for the `Person`.

Availability

IBM Security Identity Manager 7.0

Synopsis

person`.getRemovedRoles(boolean convertReplaceToAddDelete)`

Description

This method defined on the person object returns an array of static roles from which the person was removed. The return type is an array of `DirectoryObjects`.

Depending on how the person modification is triggered, the role changes get stored as `REPLACE` operation or `ADD/DELETE` operation.

When the method is invoked with a true parameter, the `REPLACE` operation is evaluated as `ADD/DELETE` and the method will return only the removed roles.

When the method is invoked with a false parameter, if role changes are stored as `REPLACE` operation, the method will return the complete set of existing roles. This return includes the existing roles, which are not deleted.

Note: The person object is often a runtime object in memory, and these static roles were not removed from the directory.

Usage `var removedRoles = per.getRemovedRoles(true);`

Person.isInRole()

The method evaluates whether a Person belongs to a role.

Availability
IBM Security Identity Manager 7.0

Synopsis
person.isInRole(*roleName*)

Arguments

roleName
The name of the role to check.

Description
Given a person object and the name of the role, determine whether the person belongs to the role. If the role is not uniquely determined by the roleName parameter or if the person cannot be found, then return an error object.

Usage

```
// Check whether the person is in the role Manager and log a
// message
var per=person.get();
if(!per.isInRole("Manager")) {
   Enrole.log("script",per.getProperty("cn")[0] +
      "does not belong to role Manager");
} else {
   Enrole.log("script",per.getProperty("cn")[0] +
      "belong to role Manager");
}
```

Person.removeRole()

The method removes the person from the specified role.

Availability
IBM Security Identity Manager 7.0

Synopsis
person.removeRole(*role*)

Arguments

role Role object that represents the role from which the person is removed.

Description
Removes the person from the role.

Note: This operation removes only the role from the Person object in run time, and it does not remove the role from the directory.

Usage

```
//Remove the first role in the Person object
var roles = person.getRoles();
if (roles.length > 0) {
   person.removeRole(roles[0]);
}
```

Person.removeRoleAssignmentData()

The method removes all role assignment data of the person for an array of assigned Roles. It does not directly change data in the data source, but removes from memory the data inside the person object.

Availability
IBM Security Identity Manager 7.0

Synopsis

person.removeRoleAssignmentData(String [] roleAssignedDNs)

Arguments

roleAssignedDNs
An array of distinguished names of the assigned role.

Description

This method is defined on the Person object. It removes all role assignment data of the person for an array of assigned roles.

Usage

```
//The script is used in a workflow, in which Entity is a person object.
var person = Entity.get();
var roleDNs = person.getProperty("erroles");
if(roleDNs.length == 0) {
 Enrole.log("script", person.name + " does not have any roles");
 return;
}

//remove the role assignment attribute.
person.removeRoleAssignmentData(roleDNs);
```

Person.updateRoleAssignmentData()

The method updates a person with the role assignment attribute value changes that are defined in the set of RoleAssignmentObjects. It does not directly change data in the data source, but updates (in memory) the data inside the person object.

Availability

IBM Security Identity Manager 7.0

Synopsis

person.updateRoleAssignmentData(RoleAssignmentObject [] roleAssignmentObject)

Arguments

roleAssignmentObject
A list of roleAssignmentObjects that contains the role assignment attribute value change set to be applied.

Description

This method is defined on the Person object. It updates a person with the role assignment attribute value changes that are defined in the set of RoleAssignmentObjects.

Usage

```
//The script is used in a workflow, in which Entity is a person object.
var person = Entity.get();
var roleDNs = person.getProperty("erroles");
if(roleDNs.length == 0) {
 Enrole.log("script", person.name + " does not have any role");
 return;
}

//construct a new RoleAssignmentObject
var assignmentObj = new RoleAssignmentObject(roleDNs[0], roleDNs[0]);
assignmentObj.addProperty("attr_3", ["newv1", "newv2"]);
person.updateRoleAssignmentData([assignmentObj]);
```

PersonSearch

The object searches for a person.

Availability

IBM Security Identity Manager 7.0
Provisioning Policy context
Service Selection Policy context

Provided by

`com.ibm.itim.script.extensions.model.PersonModelExtension`

Constructor

`new PersonSearch()`

Returns

The newly created and initialized person search object.

Methods

searchByFilter()

Search for a person by a filter.

searchByURI()

Search for a person by URI in an organizational container.

Description

The entity implements the IBM Security Identity Manager `PersonSearch` class. The API Javadoc for this class is in the following directory:

`/extensions/`*`version_number`*`/api/com/ibm/itim/dataservices/model/domain/`

Do these steps to access contents from the `extensions.zip` file:

1. Log on to the IBM Security Identity Manager virtual appliance console to open the **Appliance Dashboard**.
2. From the top-level menu of the **Appliance Dashboard**, select **Configure** > **Advanced Configuration** > **Custom File Management** to display the Custom File Management page.
3. Click the **All Files** tab.
4. Go to `directories/utilities`.
5. Select `extensions.zip` and click **Download**.
6. Extract the `extensions.zip` file.
7. Go to `/extensions/`*`version_number`*`/api/com/ibm/itim/dataservices/model/domain/`. For example, *version_number* is `7.0`.

PersonSearch.searchByFilter()

The method searches for a person by a filter.

Availability

IBM Security Identity Manager 7.0

Synopsis

`personSearch``.searchByFilter(`*`profileName`*`, `*`filter`*`, `*`scope`*`)`

Arguments

profileName

The name of the person profile to use.

filter LDAP search filter that defines the criteria for returned containers to meet. The filter must be in the format defined by RFC2254.

scope Optional search scope. Use **1** for One Level Scope and **2** for SubTree Scope. One Level Scope is the default scope.

Returns

An array of `DirectoryObjects` representing the results of the search.

Description

This method searches for a person by a filter.

Usage

```
var personSearch = new PersonSearch();
var searchResult1 = personSearch.searchByFilter("Person",
  "(sn=Smith)", 2);

// use default one level scope
var searchResult2 = personSearch.searchByFilter("Person",
    "(sn=Smith)");
```

PersonSearch.searchByURI()

The method finds a person by URI within an organizational container.

Availability

IBM Security Identity Manager 7.0

Synopsis

```
PersonSearch.searchByURI(containerDN, uri)
```

Arguments

Container DN

String representing the distinguished name of the parent organizational container.

uri String representing the URI of the person.

Returns

A Person object.

Description

Given the distinguished name of the parent organizational container and the person URI, this method finds the person. If the person is not found, this function returns null. If more than one persons found, this function throws a scripting exception.

Usage

```
var person= (new PersonSearch()).searchByURI(container.dn, uri);
if (person != null) {
Enrole.log("script", "Found " + person.getProperty("cn") );}
```

PostOffice

The object post office object that consolidates notifications.

Availability

IBM Security Identity Manager 7.0

Provided by

`com.ibm.itim.mail.postoffice.script.PostOfficeExtension`

Methods

getAllEmailMessages()
Obtains the Subject, Text Body, and HTML Body of each individual message contained in an aggregate message.

getEmailAddress()
Contains the email address that is the destination of the aggregate email message.

getPersonByEmailAddress()
Returns the Person that corresponds to the email address specified.

getTopic()
Returns the topic of the aggregated email message.

The `getAllEmailMessages()` extension allows access to the `NotificationMessage` object. Do not call the `getHtmlMessage()` method from a template. This call returns an XHTML version of the notification text. It is not possible to embed XML documents, so a call to this method results in a template execution failure. Use the text body of the original notifications by calling `getMessage()` instead.

PostOffice.getAllEmailMessages()

The message returns an array of `NotificationMessage` objects.

Availability
IBM Security Identity Manager 7.0

Synopsis
`PostOffice.getAllEmailMessages()`

Description
This JavaScript extension returns an array of `NotificationMessage` objects for obtaining the Subject, Text Body, and HTML Body of each message in an aggregate message.

Usage An example of how to iterate through the returned array in JavaScript is as follows:

```
Here are the email text bodies fetched using the JavaScript extension:
<JS>
    var msgListIterator =
      PostOffice.getAllEmailMessages().iterator();
    var returnString = "<br />";
    while (msgListIterator.hasNext()) {
        returnString = returnString +
        msgListIterator.next().getMessage() + "<br />";
    }
    return returnString;
</JS>
```

PostOffice.getEmailAddress()

The method returns email address of aggregate email destination.

Availability
IBM Security Identity Manager 7.0

Synopsis
`PostOffice.getEmailAddress()`

Description
This JavaScript extension returns a String containing the email address that is the destination of the aggregate email message.

Usage `destinationAddress = PostOffice.getEmailAddress();`

PostOffice.getPersonByEmailAddress()

The method returns the Person object that corresponds to this email address.

Availability
IBM Security Identity Manager 7.0

Synopsis
`PostOffice.getPersonByEmailAddress(`*`String email`*`)`

Description
This JavaScript extension returns the Person object that corresponds to the email address specified.

Usage `targetPerson = PostOffice.getPersonByEmailAddress()`

Examples:

```
targetPerson = PostOffice.getPersonByEmailAddress("user@itim.com");
targetPerson =
 PostOffice.getPersonByEmailAddress(PostOffice.getEmailAddress());
```

PostOffice.getTopic()

The method returns the topic string of the aggregate email.

Availability
IBM Security Identity Manager 7.0

Synopsis
`PostOffice.getTopic()`

Description
This JavaScript extension returns a string containing the topic of the aggregated email message.

Usage `topicString = PostOffice.getTopic();`

Process

Represents the IBM Security Identity Manager workflow process.

Availability
IBM Security Identity Manager 7.0

Provided by
The Process JavaScript Object in the `WorkflowExtension` returns a Process object. The object represents the current workflow process. The parent processes of the current workflow can be returned by calling `Process.getParent()` recursively, and the parent process is also a Process object.

Properties

Note: Custom result codes are supported in the workflow designer for approval activities.

APPROVED
Approved process summary code. Result code is `AA`.

ESCALATED
Escalated process summary code. Result code is `ES`.

FAILED
Failed process summary code. Result code is SF.

PARTICIPANT_RESOLVE_FAILED
Participant resolved failure process summary code. Result code is PF.

PENDING
Pending process summary code. Result code is PE.

REJECTED
Rejected process summary code. Result code is AR.

SUBMITTED
Submitted process summary code. Result code is RS.

SUCCESS
Success process summary code. Result code is SS.

TIMEOUT
Time out process summary code. Result code is ST.

WARNING
Warning process summary code. Result code is SW.

comment
Provides additional information about the process given when defined in the workflow designer.

description
Describes the purpose of the process given when defined in the workflow designer.

id Assigned by the workflow designer to uniquely identify the workflow process within the workflow engine.

name Label given this activity when defined in the workflow designer.

parentId
Uniquely identifies the parent process (if any) that started this process.

requesteeDN
Uniquely identifies the requestee if the requestee is a user in the IBM Security Identity Manager data store.

requesteeName
Name of the process requestee.

requestorId
Id of the process requestor.

requestorName
The name of the process requestor if the requestor is a user.

requestorType
Categorize the requestor

resultDetail
An application-specific string that provides more detail about the result of the process.

resultSummary
An application-specific string that represents the summary result of the process.

started
Indicates when the process started.

state Code that represents the current state of the process.

subject
Describes the object that is the focal point of the workflow process.

type Code that categorizes the process given when defined in the workflow designer.

Methods

auditEvent()
Create an event in the audit trail specific to the activity.

getActivity()
Returns an activity with the ID and index.

getParent()
Get the parent process (if any) that started this process.

getRootProcess()
Returns the JavaScript Process object that contains information about the root process.

getRootRequesterName()
Returns String of requester name of the root process.

setRequesteeData()
Change the requestee data for the current process.

setResult()
Change the result member of the activity in the current activity.

setSubjectData()
Change the subject data for the current process.

Description
This entity represents the current workflow process is running.

Process.auditEvent()

The method creates an event in the audit trail.

Availability
IBM Security Identity Manager 7.0

Synopsis
process.auditEvent(*event*)

Arguments

event String representing the event to be audited.

Description
This method creates an event in the audit trail specific to the process. The function takes in one parameter that can be any JavaScript object that can be translated into a string for storage. In the audit trail, the event is automatically time stamped.

Usage
```
process.auditEvent("Task completed");
```

Process.comment

The field provides additional information about the process.

Availability
IBM Security Identity Manager 7.0

Synopsis
process.comment

Description
This read-only field is a string that provides additional information about the process given when defined in the workflow designer.

Usage `x = process.comment;`

Process.description

The field represents the purpose of the process.

Availability
IBM Security Identity Manager 7.0

Synopsis
process.description

Description
This read-only field is a string that describes the purpose of the process when defined in the workflow designer.

Usage `x = process.description;`

Process.getActivity()

The method returns an activity with the ID and index.

Availability
IBM Security Identity Manager 7.0

Synopsis
process.getActivity(*id*, *index*)

Arguments

id Activity ID assigned by the workflow designer.

index Optionally identifies specific activity if there is more than one activity with the ID.

Returns
The associated Activity.

Description
This method returns an activity with the ID and index in the event that there is more than one activity with the ID. This might occur if the activity of the given ID is called multiple times in a loop in the workflow process. If there is no activity with the ID and index, this function returns null. If the optional index is not specified and if there is more than one activity with the ID, the first activity with the ID is returned.

Usage

```
theFirstActivity = process.getActivity("id1", 3);
theActivityName = theFirstActivity.name;

theSecondActivity = process.getActivity("id2");
theActivityName = theSecondActivity.name;
```

Process.getParent()

The method returns the parent process (if any) that started this process.

Availability
IBM Security Identity Manager 7.0

Synopsis
process.getParent()

Returns
The parent Process. If there is no parent, a null is returned.

Description
This method returns the parent process (if any) that started this process.

Usage

```
parent = process.getParent();
parentName = parent.name;
```

Process.getRootProcess()

The method returns the root process (if any) that started this process.

Availability

IBM Security Identity Manager 7.0

Synopsis

```
process.getRootProcess()
```

Returns

The root process. If there is no root process, a null is returned.

Description

This method returns the root process (if any) of this process.

Usage

```
root = process.getRootProcess();
rootName = root.name;
```

Process.getRootRequesterName()

The method returns the root requester name.

Availability
IBM Security Identity Manager 7.0

Synopsis
process.getRootRequesterName()

Description
This method returns the root requester name of the workflow process initiator.

Usage `rootRequester = process.getRootRequesterName();`

Process.guid

The generated unique identifier assigned to the process at runtime.

Availability
IBM Security Identity Manager 7.0

Synopsis
process.guid

Description
This read-only field is a String of the generated unique identifier for the workflow process in the workflow engine.

Usage `x = process.guid;`

Process.getSubProcesses()

The method returns the subordinate processes (if any) of the process.

Availability

IBM Security Identity Manager 7.0

Synopsis

```
process.getSubProcesses()
```

Returns

The subordinate processes. If there are no subordinate processes, an empty array is returned.

Description

This method returns the subordinate processes (if any) of this process.

Usage

```
var out = "subprocesses of the process: \n";

function traverse(p, prefix) {
  var subProcesses = p.getSubProcesses();
  prefix += "/" + p.name;
  out += prefix + ": " + p.id + " type: " + p.type + " resultSummary: " + p.resultSummary + "\n";
  for (var i = 0; i < subProcesses.length; i++) {
      traverse(subProcesses[i], prefix);
  }
}

traverse(process, "");
activity.auditEvent(out);
```

Process.id

The generated unique identifier assigned to the process at runtime.

Availability
IBM Security Identity Manager 7.0

Synopsis
process.id

Description
This read-only field is a string of the generated unique identifier for the workflow process in the workflow engine.

Usage `x = process.id;`

Process.name

The label assigned to the process.

Availability
IBM Security Identity Manager 7.0

Synopsis
`process.name`

Description
This read-only field is a string assigned by the workflow designer to label this process.

Usage `x = process.name;`

Process.parentId

The field uniquely identifies the parent process that started this process.

Availability
IBM Security Identity Manager 7.0

Synopsis
`process.parentId`

Description
This read-only field is a string representation of the long integer that uniquely identifies the parent process (if any) that started this process.

Usage `x = process.parentId;`

Process.requesteeDN

The field uniquely identifies the requestee if the requestee is a user in the IBM Security Identity Manager data store.

Availability
IBM Security Identity Manager 7.0

Synopsis
`process.requesteeDN`

Description
This read-only field is a string that uniquely identifies the requestee if the requestee is a user in the IBM Security Identity Manager data store. Not all requestees are users (that is, the process can act on a policy, not a user directly), so this member can be empty.

Usage `x = process.requesteeDN;`

Process.requestorDN

The field specifies the distinguished name of the process requester, if the requester is a user in the IBM Security Identity Manager data store.

Availability
IBM Security Identity Manager 7.0

Synopsis

process.requestorDN

Description

This read-only field is a string that represents the distinguished name of the process requester. This string is displayed only if the requester is a user in the IBM Security Identity Manager data store. Not all requesters are users (that is, the process can act on a policy, not a user directly), so this member can be empty.

Usage

```
if (process.requestorType == "U")
x = process.requestorDN;
```

Process.requesteeName

The field represents the name of the process requestee as a string.

Availability

IBM Security Identity Manager 7.0

Synopsis

process.requesteeName

Description

This read-only field is a string that provides the name the requestee if the requestee is a user in the IBM Security Identity Manager data store. Not all requestees are users (that is, the process can act on a policy, not a user directly), so this member can be empty.

Usage

```
x = process.requesteeName;
```

Process.requestorName

The field represents the name of the process requester if the requester is a user.

Availability

IBM Security Identity Manager 7.0

Synopsis

process.requestorName

Description

This read-only field is a string that represents the name of the process requester if the requester is a user.

Usage

```
if (process.requestorType == "U")
  x = process.requestorName;
```

Process.requestorId

The field identifies the party who requested the process to be executed.

Availability

IBM Tivoli Identity Manager 4.x

Synopsis

process.requestorId

Description

This read-only field is a String that identifies the party who requested the process to be executed. The value is not guaranteed to be unique across different workflow processes.

Usage `processRequestorId = process.requestorId;`

Note: If the requestor is system (`process.requestorType == P`) then `process.requestorId` returns `NULL`.

Process.requestorType

The field categorize the requestor.

Availability
IBM Security Identity Manager 7.0

Synopsis
`process.requestorType`

Description
This read-only field is a string that categorizes the requestor. The potential categories, or types, are:

- **U** for user
- **S** for the workflow engine
- **P** for the system

Usage

```
x = process.requestorType;
if (x == "U")
  ...
else if (x == "S")
  ...
else if (x == "P")
  ...
```

Process.resultDetail

The field details about the result of the process.

Availability
IBM Security Identity Manager 7.0

Synopsis
`process.resultDetail`

Description
This read-only field is an application-specific string that provides more detail about the result of the process.

Usage `x = process.resultDetail;`

Process.resultSummary

The field represents the summary of the result of the process.

Availability
IBM Security Identity Manager 7.0

Description
This read-only field is an application-specific string that provides a summary of the result of the process.

Usage `x = process.resultSummary;`

Process.setRequesteeData()

The method changes the requestee data for the current process.

Availability
IBM Security Identity Manager 7.0

Synopsis
process.setRequesteeData(*person*)

Arguments

person
`DirectoryObject` representing the new requestee.

Description
This method changes the requestee data for the current process. It is not supported for a process that is not the current process. It not only updates the current process in the script, but also in the workflow engine. The `requesteeData` argument contains a person distinguished name or a collection of strings from which the requestee data can be extracted.

Usage *process*.setRequesteeData(*person*);

Process.setResult()

The method changes the result member of the process.

Availability
IBM Security Identity Manager 7.0

Synopsis
process.setResult(*summary*, *detail*)

Arguments

summary
String code that represents the result summary.

detail String representing the result details.

Description
This method changes the result member of the process in the current process. It is supported for current activities in the current workflow process. The result is composed by an application-specific summary code, and optional more detailed application-specific description. The summary code can indicate a success or failure. This summary code is stored as the `resultSummary` member locally and updated in the relevant data in the workflow engine. The detail is stored as the `resultDetail` member locally and updated in the relevant data in the workflow engine.

Usage

```
process.setResult(process.FAILED, "Unable to connect to resource");
```

Process.setSubjectData()

The method changes the subject data for the current process.

Availability
IBM Security Identity Manager 7.0

Synopsis
process.setSubjectData(*person*)

Arguments

person
`DirectoryObject` representing the new subject.

Description

This method changes the subject data for the current process. It is not supported for a process that is not the current process. It not only updates the current process in the script, but also in the workflow engine. The `subjectData` argument contains a person distinguished name or a collection of strings from which the subject data can be extracted.

Usage *process*`.setSubjectData(`*person*`);`

Process.started

The field represents the JavaScript date that indicates when the process started.

Availability

IBM Security Identity Manager 7.0

Synopsis

process`.started`

Description

This read-only field is code string that represents the JavaScript Date that indicates when the process started.

Usage

```
x = process.started;
```

Process.state

The field represents the current state of the process.

Availability

IBM Security Identity Manager 7.0

Synopsis

process`.state`

Description

This read-only field is code string that represents the current state of the process. The state can have the following values:

- `R` for running
- `I` for not started
- `T` for terminated
- `A` for aborted
- `S` for suspended
- `C` for completed
- `B` for bypassed

Usage

```
if (process.state == "S") {
  ...
}
```

Process.subject

The field represents the object that is the focal point of the workflow process.

Availability

IBM Security Identity Manager 7.0

Synopsis

process`.subject`

Description

This read-only field is code string that describes the object that is the focal point of the workflow process. This string can be an identity in the system, an account, a policy, or another object.

Usage `x = process.subject;`

Process.type

The field represents the type of process.

Availability

IBM Security Identity Manager 7.0

Synopsis

`process.type`

Description

This read-only field is code string that categorizes the process when defined in the workflow designer.

Usage `x = process.type;`

ProcessData

The object represents the workflow process data entity.

Availability

IBM Security Identity Manager 7.0
Workflow context

Provided by

com.ibm.itim.workflow.script.WorkflowExtension

Methods

get() Returns a JavaScript object that represents the value of the relevant data item.

set() Changes the value of the relevant data item.

Description

Each workflow process has a set of relevant data, or process specific parameters, which can be read or changed from within a workflow script. The name and syntax of these parameters, or relevant data items, are defined in the workflow designer, and are typically specific to the workflow process purpose. For example, when adding a user, an object that holds all the attributes of the new user can be a relevant data item. However, when deleting a user, the only needed relevant data item can be the distinguished name of the user to delete.

Each relevant data item will be represented in the workflow script as a variable with the same relevant data ID as defined in the workflow designer.

ProcessData.get()

The method changes the subject data for the current process.

Availability

IBM Security Identity Manager 7.0

Synopsis

processData.get()

Returns

Returns a JavaScript object that represents the value of the relevant data item.

Description

This method returns a JavaScript object that represents the value of the relevant data item. There is a variable present for each relevant data item in the context of script. For performance reasons, the values are not retrieved from the workflow engine until the script specifically requests the values with this call. The returned JavaScript object is in the same syntax as defined in the workflow designer.

Usage

```
dn = subjectDN.get();
```

ProcessData.set()

The method changes the value of the relevant data item.

Availability

IBM Security Identity Manager 7.0

Synopsis

processData.set(value)

Arguments

value Value to use to update the relevant data item.

Description

This method changes the value of the relevant data item. It not only updates the relevant data item in the script, but also in the workflow engine. The new value is a parameter to the function. The new value must be compatible with the syntax of the relevant data item as defined in the workflow designer. For example, if the relevant data item is an integer, the value cat would not be a valid parameter to this function.

Usage

```
processData.set("engineering");
```

RecertificationWorkflow

Provides extended capabilities to user recertification workflows, including audit support for the reporting and view requests functions.

Availability

IBM Security Identity Manager 7.0.

Methods

auditCompletion(person, recertPolicy, approvalDoc)

Performs a full audit of the decisions made during a user recertification packaged approval activity, including data for reporting and view requests.

auditTimeout(person, recertPolicy, approvalDoc)

Perform full audit of the decisions set during a user recertification packaged approval activity timeout, including data for reporting and view requests.

auditCompletion(person, recertPolicy, approvalDoc, auditForReports, auditForViewRequests)
Performs an audit of the decisions made during a user recertification packaged approval activity. The value of the Boolean flag `auditForReports` determines whether an audit entry is created for reporting. The value of the Boolean flag `auditForViewRequests` determines whether an audit entry is created for view requests.

RecertificationWorkflow.auditTimeout(person, recertPolicy, approvalDoc, auditForReports, auditForViewRequests)
Performs an audit of the decisions set during a user recertification packaged approval activity timeout. The value of the Boolean flag `auditForReports` determines whether an audit entry is created for reporting. The value of the Boolean flag `auditForViewRequests` determines whether an audit entry is created for view requests.

Usage
```
RecertificationWorkflow.auditCompletion(Entity.get(), Policy.get(),
ApprovalDocument.get())
```

```
RecertificationWorkflow.auditTimeout(Entity.get(), Policy.get(),
ApprovalDocument.get(), false, true)
```

Reminder

An activity to-do item reminder informs the participant that the IBM Security Identity Manager requires user action.

Availability

IBM Security Identity Manager 7.0
Reminder context

Provided by
`com.ibm.itim.script.extensions.ReminderExtension`

Methods

Reminder.getOriginalSubject()
This method returns the subject of the original notification sent when the work item was first assigned.

Reminder.getXhtmlBody()
This method returns the XHTML body of the original notification sent when the work item was first assigned.

Reminder.getTextBody()
This method returns the text body of the original notification sent when the work item was first assigned.

Reminder.getRemindersSent()
This method returns the number of reminders previously sent.

Reminder.getEscalationTime()
This method returns a string that contains the date and time when the work item is escalated unless acted upon.

Reminder.getEscalationDate()
This method returns a Date containing the date and time when the work item is escalated unless acted upon.

Description

An activity to-do item reminder informs the participant that IBM Security Identity Manager requires user action.

Role

The object represents the role associated with a provisioning operation.

Availability

IBM Security Identity Manager 7.0

Provided by

`com.ibm.itim.script.extensions.model.RoleModelExtension`

Constructor

new `Role(dn)`

Returns

A new Role object that represents the Role with the given DN.

Methods

getAssignmentAttributes()

Returns an array of assignment attribute names. Returns an empty array if no assignment attribute exists.

getAllAssignmentAttributes()

Returns an array of `RoleAssignmentAttribute` objects containing assignment attribute name, role name, and role DN. Returns an empty array if no assignment attribute exists. Returns the role assignment attributes of the whole role hierarchy.

getOwner()

Returns an array of `DirectoryObjects` that represent any Person that has an Owner relationship with this role.

getChildRoles()

Returns an array of `roles`. The array contains the immediate member roles, that is, child roles of the role. The method returns an empty array if no child role exists.

getParentRoles()

Returns an array of `roles`. The array contains the immediate parent roles of the role. The method returns an empty array if no parent role exists.

getAscendantRoles()

Returns an array of `roles`. The array contains all ancestor roles of the role, transitively. The method returns an empty array if no ascendant role exists.

getDecendantRoles()

Returns an array of `roles`. The array contains all member roles of the role, transitively, based on the role hierarchy. The method returns an empty array if no member role exists.

setAssignmentAttributes()

Sets role assignment attributes of the role.

Inherits from

`DirectoryObject`

Synopsis

```
role.dn;
```

Description

The role object is available in the context of a provisioning policy.

Note: For more information on role assignment attributes, see Defining assignment attributes when creating a role.

Role.getAscendantRoles()

This method returns all the ascendant roles, transitively. Ascendant roles are ancestors, that is, parent roles, grandparent roles, and more remote ancestor roles.

Availability

IBM Security Identity Manager 7.0.

Synopsis

```
Role.getAnscendantRoles()
```

Arguments

None

Description

This method is defined on the Role object and returns an array of roles. The array contains all ancestor roles of the role, transitively. The method returns an empty array if no ascendant role exists.

Usage

```
var role = new Role(roleDN);
//get ascendant roles
var roleList = role.getAscendantRoles();
if (roleList.length == 0) {
    Enrole.log("script", "There are no ascendant roles of role: " + role.name);
    return;
}

// print out role names.
for (var i=0; i < roleList.length; i++) {
    var r = roleList[i];
    Enrole.log("script","role name: "+ r.name);
}
```

Role.getAssignmentAttributes()

The method returns an array of assignment attribute names. Returns an empty array if no assignment attribute exists.

Availability

IBM Security Identity Manager 7.0.

Synopsis

```
Role.getAssignmentAttributes()
```

Arguments

None

Description

This method is defined on the Role object and returns an array of assignment attribute names. The method returns an empty array if no assignment attribute exists.

Usage

```
var role = new Role(roleDN);

//get assignment attributes of the role
var attributeList = role.getAssignmentAttributes();
if (attributeList.length == 0) {
    Enrole.log("script", "No assignment attribute for this role: "
        + role.name);
    return;
}

// print out role assignment attribute name.
for (var i=0; i < attributeList.length; i++) {
    var attrName = attributeList[i];
    Enrole.log("script","attribute name-----: "+ attrName);
}
```

Role.getChildRoles()

The method returns all the immediate member roles.

Availability

IBM Security Identity Manager 7.0.

Synopsis

```
Role.getChildRoles()
```

Arguments

None

Description

This method is defined on the Role object and returns an array of roles. The array contains the immediate member roles, that is, child roles of the role. The method returns an empty array if no child role exists.

Usage

```
var role = new Role(roleDN);
//get child roles
var roleList = role.getChildRoles();
if (roleList.length == 0) {
    Enrole.log("script", "There are no child roles of role: " + role.name);
    return;
}

// print out role names.
for (var i=0; i < roleList.length; i++) {
    var r = roleList[i];
    Enrole.log("script","role name: "+ r.name);
}
```

Role.getDescendantRoles()

This method returns all the member roles, transitively.

Availability

IBM Security Identity Manager 7.0.

Synopsis

```
Role.getDescendantRoles()
```

Arguments

None

Description

This method is defined on the Role object and returns an array of roles.

The array contains all member roles of the role, transitively, based on the role hierarchy. The method returns an empty array if no member role exists.

Usage

```
var role = new Role(roleDN);
//get child roles
var roleList = role.getDescendantRoles();
if (roleList.length == 0) {
    Enrole.log("script", "There are no descendant roles of role: " + role.name);
    return;
}

// print out role names.
for (var i=0; i < roleList.length; i++) {
    var r = roleList[i];
    Enrole.log("script","role name: "+ r.name);
}}
```

Role.getOwner()

The method returns an array of `DirectoryObjects` that represents any Person that has an Owner relationship with this role.

Availability

IBM Security Identity Manager 7.0

Synopsis

```
Role.getOwner()
```

Returns

Array of `DirectoryObjects` that represents the owners of this Role or null if there are no owners.

Usage

```
var owners = role.getOwner();
```

Role.getParentRoles()

The method returns all the immediate parent roles.

Availability

IBM Security Identity Manager 7.0.

Synopsis

```
Role.getParentRoles()
```

Arguments

None

Description

This method is defined on the Role object and returns an array of roles. The array contains the immediate parent roles of the role. The method returns an empty array if no parent role exists.

Usage

```
var role = new Role(roleDN);
//get parent roles
var roleList = role.getParentRoles();
if (roleList.length == 0) {
    Enrole.log("script", "There is no parent role of role: " + role.name);
    return;
}

// print out role names.
```

```
for (var i=0; i < roleList.length; i++) {
    var r = roleList[i];
    Enrole.log("script","role name: "+ r.name);
}
```

Role.setAssignmentAttributes()

The method sets role assignment attributes of the role.

Availability

IBM Security Identity Manager 7.0.

Synopsis

```
Role.setAssignmentAttributes(String[] attributeNames)
```

Arguments

attributeNames

The array of assignment attribute names of the role. If an empty array is specified, all assignment attributes for the role are removed.

Description

This method is defined on the Role object and sets the role assignment attributes for a role.

Usage

```
var roleDN = roles[0];
var role = new Role(roleDN);
var roleAtr = new Array();
roleAtr[0] = "creditlimit";
//set assignment attribute names
role.setAssignmentAttributes(roleAtr);
```

RoleAssignmentAttribute

The object represents the role assignment attribute associated with a role.

Availability

IBM Security Identity Manager 7.0.

Methods

getName()

Returns the attribute name associated with the role assignment attribute object.

getRoleName()

Returns the name of the role. Returns an empty string if there is no name associated with the role assignment attribute object.

getRoleDN()

Returns the DN of the role. Returns an empty string if there is no DN associated with the role assignment attribute object.

Description

The `RoleAssignmentAttribute` object associated with the role assignment attribute.

RoleAssignmentAttribute.getName()

The method returns the name of the assignment attribute.

Availability

IBM Security Identity Manager 7.0.

Synopsis

```
RoleAssignmentAttribute.getName()
```

Arguments

None

Returns

The name of the assignment attribute.

Description

Returns the name of the assignment attribute that is defined on the role.

Usage

```
var role = new Role(roleDN);
    //get assignment attributes of the role
    var attributeList = role.getAllAssignmentAttributes();
    if (attributeList.length == 0) {
        Enrole.log("script", "No assignment attribute for this role: "
            + role.name);
        return;
    }

    // print out role assignment attribute name.
    for (var i=0; i < attributeList.length; i++) {
        var roleAtr = attributeList[i];
        Enrole.log("script","attribute name-----: "+ roleAtr.getName());
    }
```

RoleAssignmentAttribute.getRoleName()

The method returns the name of the role that has the assignment attribute defined.

Availability

IBM Security Identity Manager 7.0.

Synopsis

```
RoleAssignmentAttribute.getRoleName()
```

Arguments

None

Returns

The name of the role that has the assignment attribute defined.

Description

Returns the name of the role that has the assignment attribute defined.

Usage

```
var role = new Role(roleDN);
    //get assignment attributes of the role
    var attributeList = role.getAllAssignmentAttributes();
    if (attributeList.length == 0) {
        Enrole.log("script", "No assignment attribute for this role: "
            + role.name);
        return;
    }

    // print out all role names.
    for (var i=0; i < attributeList.length; i++) {
        var roleAtr = attributeList[i];
        Enrole.log("script","role name-----: "+ roleAtr.getRoleName());
    }
```

RoleAssignmentAttribute.getRoleDN

The method returns the distinguished name of the role that defines the assignment attributes.

Availability

IBM Security Identity Manager 7.0.

Synopsis

```
RoleAssignmentAttribute.getRoleDN()
```

Arguments

None

Returns

The distinguished name of the role that defines the assignment attributes.

Description

Returns the distinguished name of the role that defines the assignment attributes.

Usage

```
var role = new Role(roleDN);
    //get assignment attributes of the role
    var attributeList = role.getAllAssignmentAttributes();
    if (attributeList.length == 0) {
        Enrole.log("script", "No assignment attribute for this role: "
            + role.name);
        return;
    }

    // print out the distinguished name of the role that defines
    // assignment attributes.
    for (var i=0; i < attributeList.length; i++) {
        var roleAtr = attributeList[i];
        Enrole.log("script","define role DN-----: "+ roleAtr.getRoleDN());
    }
```

RoleAssignmentObject

The `RoleAssignmentObject` class is a DataObject class for role assignment data.

This class holds the assignment data that are associated with the defined role and the assigned role. The defined role is the role that holds a list of assignment attributes. The assigned role is the role to which the person is assigned.

Availability

IBM Security Identity Manager 7.0

Provided by

`com.ibm.itim.script.extensions.model.RAObjectModelExtension`

Constructors

new RoleAssignmentObject(RoleAssignmentObject assignmentObject)

Arguments:

assignmentObject

`RoleAssignmentObject` that is wrapped inside the `RoleAssignmentObject`.

new RoleAssignmentObject(String assignedRoleDN, String definedRoleDN)

Arguments:

assignedRoleDN
The String format of the distinguished name for the assigned role.

definedRoleDN
The String format of the distinguished name for the defined role.

Methods

addProperty()
Adds the values for specified assignment attribute.

getAssignedRoleDN()
Returns the distinguished name string for the role to which the person is assigned.

getDefinedRoleDN()
Returns the distinguished name string for the role in which the assignment attribute is defined.

getChanges()
Returns the changes made to this `RoleAssignmentObject`.

getProperty()
Returns the values of the property specified by the assignment attribute name.

getPropertyNames()
Returns a list of role assignment attribute names.

removeProperty()
Removes the values for the specified assignment attribute name.

setProperty()
Sets the values for a specified assignment attribute.

Description
RoleAssignmentObject contains the role assignment data, including the assigned role DN, the defined role `DN` and attribute values.

RoleAssignmentObject.getAssignedRoleDN()

The method returns the distinguished name string for the role to which a person is assigned.

Availability
IBM Security Identity Manager 7.0.

Synopsis
`roleAssignmentObject``.getAssignedRoleDN()`

Arguments
None

Returns
The distinguished name string for the role to which a person is assigned.

Description
This method returns the distinguished name string for the role to which a person is assigned.

Usage

```
var assignedRoleDN = "globalid=111";
var definedRoleDN = "globalid=222";
var assignmentObj = new RoleAssignmentObject(assignedRoleDN, definedRoleDN);

var assignedRoleDN2 =  assignmentObj.getAssignedRoleDN();
```

RoleAssignmentObject.getDefinedRoleDN()

The method returns the distinguished name string for the role in which the assignment attribute is defined.

Availability

IBM Security Identity Manager 7.0.

Synopsis

roleAssignmentObject.getDefinedRoleDN()

Arguments

None

Returns

Returns the distinguished name string for the role in which the assignment attribute is defined.

Description

This method returns the distinguished name string for the role to which the person is assigned.

Usage

```
var assignedRoleDN = "globalid=111";
var definedRoleDN = "globalid=222";
var assignmentObj = new RoleAssignmentObject(assignedRoleDN, definedRoleDN);

var definedRoleDN2 =  assignmentObj.getDefinedRoleDN();
```

RoleAssignmentObject.addProperty()

Use this method to add the values for specified assignment attribute.

Availability

IBM Security Identity Manager 7.0.

Synopsis

RoleAssignmentObject.addProperty(*name*, *value*)

Arguments

name String representing the name of the assignment attribute to be added.

value The value to be added.

Description

This method changes the value of the specified assignment attribute or adds the specified assignment attribute if it does not exist. This change is made locally to the script environment, not to the data store.

Usage

```
// Create assignment object with assigned role dn and defined role dn.
var assignmentObj = new RoleAssignmentObject("eruid=1111,dc=com",
    "eruid=2222,dc=com");
// Add some assignment attribute with values.
assignmentObj.addProperty("attr1", ["attr1val1","attr2val1"]);
assignmentObj.addProperty("attr2", ["attr2val1"]);
assignmentObj.addProperty("attr2", ["attr2val2"]);
```

RoleAssignmentObject.getChanges()

The method returns the changes made to an entity.

Availability

IBM Security Identity Manager 7.0.

Synopsis

RoleAssignmentObject.getChanges()

Returns

An array of change objects. If there are no changes, an empty array is returned. Each element in the array is an AttributeChangeOperation.

Description

This method returns the changes made to the entity. These changes are represented by change objects with the following members:

attr String name of the attribute that is being changed.

op An integer that identifies the type of change that is being made. The enumerated values are 1 for add, 2 for replace, and 3 for remove.

values An array of objects that can be either added, removed, or replaced.

The changes are returned as an array of these change objects. If there are no changes, an empty array is returned.

Usage

```
changes = assignmentObject.getChanges();
for (i = 0; i < changes.length; i++) {
  name = changes[i].attr;
  if (changes[i].op == 1) {
    ...
  } else if (changes[i].op == 2) {
    ...
  } else {
    ...
  }
};
```

RoleAssignmentObject.getProperty()

The method returns the values of the assignment attribute specified by the given name.

Availability

IBM Security Identity Manager 7.0.

Synopsis

RoleAssignmentObject.getProperty(*name*)

Arguments

name String representing the name of the assignment attribute to return.

Returns

The array of strings that represents the values for an assignment attribute. If the specified assignment attribute does not exist, an empty array is returned.

Description

This method returns the values of the assignment attribute specified by the given name. If the specified assignment attribute does not exist, an empty array is returned.

Usage

```
// create assignment object with assigned role dn and defined role dn.
var assignmentObj = new RoleAssignmentObject("eruid=1111,dc=com",
    "eruid=2222,dc=com");
assignmentObj.addProperty("attr1", ["attr1val1", "attr1val2"]);

// get assignment attribute values for attr1.
var attrValues = assignmentObj.getProperty("attr1");
var attrValuesStr = "";
for (var j=0; j<attrValues.length; j++) {
  attrValuesStr += attrValues[j] + ", ";
Enrole.log("script", "The values for attr1:" + attrValuesStr);
```

RoleAssignmentObject.getPropertyNames()

The method returns a list of assignment attributes.

Availability

IBM Security Identity Manager 7.0.

Synopsis

RoleAssignmentObject.getPropertyNames()

Returns

An array of strings.

Description

This method returns a list of assignment attributes as an array of strings.

Usage `properties = RoleAssignmentObject.getPropertyNames();`

RoleAssignmentObject.removeProperty()

The method removes the assignment attribute specified by the given name.

Availability

IBM Security Identity Manager 7.0.

Synopsis

RoleAssignmentObject.removeProperty(*name*)

Arguments

name String representing the name of the assignment attribute to remove.

Description

This method removes the specified assignment attribute. This change is made locally to the script environment, not to the data store.

Usage `RoleAssignmentObject.removeProperty("assignmentAttr1");`

RoleAssignmentObject.setProperty()

The method sets the value of the specified assignment attribute.

Availability

IBM Security Identity Manager 7.0.

Synopsis

RoleAssignmentObject.setProperty(*name*, *value*)

Arguments

name String representing the name of the assignment attribute to be created or modified.

value Specifies the value to which the assignment attribute is set.

Description

This method changes the value of the specified assignment attribute, or adds the specified assignment attribute if it does not exist. This change is made locally to the script environment, not to the data store.

Usage *RoleAssignmentObject*.setProperty("attr1",["val1","val2"]);

RoleSearch

The object searches for a role.

Availability

IBM Security Identity Manager 7.0

Provided by

com.ibm.itim.script.extensions.model.RoleModelExtension

Constructor

new RoleSearch()

Returns

The newly created and initialized role search object.

Methods

searchByName()

Search for a role by name.

searchByURI()

Search for a role by URI within an organizational container.

RoleSearch.searchByName()

The method searches for a role by a name.

Availability

IBM Security Identity Manager 7.0

Synopsis

RoleSearch.searchByName(*name*)

Arguments

name The role name to use as the basis for the search.

Returns

Array of DirectoryObjects that represents a role.

Description

Given the name of a role, locate the Role entity. Will return null if there is not exactly one matching role.

Usage

```
// Given the name of a role, see if it exists and log its
// description
var roles = (new RoleSearch()).searchByName("testRole");
if (roles.length >= 1) {
  if (roles[0].getProperty("errolename")[0] == "testRole") {
```

```
   Enrole.log("script", "The Role "+ roles[0].getProperty("errolename")[0] +
    "has Description :" + roles[0].getProperty("description")[0]);
  }
}
```

RoleSearch.searchByURI()

The method finds a role by URI in an organizational container.

Availability

IBM Security Identity Manager 7.0.

Synopsis

```
RoleSearch.searchByURI(containerDN, uri)
```

Arguments

Container DN
String representing the distinguished name of the organizational container.

uri String representing the URI of the role.

Returns

A Role object

Description

Given the distinguished name of the organizational container and the role URI, this method finds the container. If the role is not found, this function returns null. If more than one role is found, this function throws a scripting exception.

Usage

```
var role = (new RoleSearch()).searchByURI(container.dn, uri);
if (role != null) {
Enrole.log("script", "Found " + role.getProperty("errolename") );}
```

SeparationOfDutyRuleViolation

Object that provides information about a specific separation of duty rule violation. Use this object to get specific information about a separation of duty policy violation. This object cannot be created for use by the user. The user can work only with `SeparationOfDutyRuleViolation` objects that the system has generated as part of the *approveSoDViolation* workflow.

Availability

IBM Security Identity Manager 7.0.

Provided by

`com.ibm.itim.script.wrappers.generic.IRuleResultWrapper`

Methods

getName()
Returns the name of the separation of duty policy rule to which this violation corresponds.

getViolationString()
Provides a string that represents the list of roles in violation. It describes the roles the person has that are in violation and which role in a separation of duty rule they correspond to. The role lists might be different due to role hierarchy.

getViolationStringHTMLTable()
Returns a string version of the roles in violation for use in an HTML table or email template.

getPolicyName()
Returns the name of the separation of duty policy which contains the rule in violation.

getPolicyDescription()
Returns a description of the separation of duty policy.

getPolicyOwnerDNs()
Returns a collection of the distinguished names of one or more separation of duty policy owners.

getCardinality()
Returns string that represents the number of allowed roles in the separation of duty policy rule in violation.

Service

The object represents the service associated with a provisioning operation.

Availability

IBM Security Identity Manager 7.0

Provided by
`com.ibm.itim.script.extensions.model.ServiceModelExtension`

Constructor
`new Service(dn)`

Returns
A new Service object that represents the Service with the DN.

Inherits From
`DirectoryObject`

Synopsis
```
service.dn;
```

Description
The service object is available in the context of a Provisioning Policy and Service Selection Policy.

ServiceSearch

Use the object to provide searching capability for IBM Security Identity Manager services.

Availability

IBM Security Identity Manager 7.0
Provisioning Policy context
Service Selection Policy context

Provided by
`com.ibm.itim.script.extensions.model.ServiceModelExtension`

Methods

searchByFilter()
Search for a service by a filter.

searchByName()
Search for a service by a name.

searchByURI()
Search for a service by URI in an organizational container.

searchForClosestToPerson()
Search for the closest Service to a person.

Description
This object is used to provide searching capability for IBM Security Identity Manager services.

ServiceSearch.searchByFilter()

The method searches for a service by a filter.

Availability
IBM Security Identity Manager 7.0

Synopsis
ServiceSearch.searchByFilter(*filter*, *scope*)

Arguments

filter LDAP search filter that defines the criteria for returned containers to meet. The filter must be in the format defined by RFC2254.

scope Optional search scope. Use 1 for One Level Scope and 2 for SubTree Scope. One Level Scope is the default scope.

Returns
An array of `DirectoryObjects` representing the results of the search.

Description
This method searches for a service by a filter.

Usage

```
searchResult1 =
 ServiceSearch.searchByFilter("(erntlocalservername=*srv)", 2);

// use default one level scope, put statement on one line

searchResult2 =
 ServiceSearch.searchByFilter("(erntlocalservername=*srv)");
```

ServiceSearch.searchByName()

The method searches for a service by name.

Availability
IBM Security Identity Manager 7.0

Synopsis
ServiceSearch.searchByName(*name*, *profileName*, *scope*)

Arguments

name The service name, provided as a string, to use as the basis for the search.

profileName
Optional profile name, provided as a string. The profile name of the service to use as the basis for the search.

scope Optional search scope, provided as an int. Use **1** for One Level Scope and **2** for Scope. One Level Scope is the default scope. When you use this method in workflow JavaScripts, set the scope parameter to SubTree because the logical search context is limited to the tenant above the default organization. In this context, setting the scope to One Level Scope returns empty results during a search because there are no services at the tenant level.

Returns
An array of `DirectoryObjects` representing the results of the search.

Description
This method searches for a service by a name.

Usage

```
searchResult1 = ServiceSearch.searchByName("US Service", 2);

// use default one level scope
searchResult2 = ServiceSearch.searchByName("US Service");
```

ServiceSearch.searchByURI()

The method finds a service by URI in an organizational container.

Availability
IBM Security Identity Manager 7.0.

Synopsis

```
ServiceSearch.searchByURI(containerDN, uri)
```

Arguments

Container DN
String representing the distinguished name of the organizational container.

uri String representing the URI of the service.

Returns
A Service object

Description
Given the distinguished name of the organizational container and the service URI, this method finds the service. If the service is not found, this function returns null. If more than one service is found, this function throws a scripting exception.

Usage

```
var service = (new ServiceSearch()).searchByURI(container.dn, uri);
if (service != null) {
Enrole.log("script", "Found " + service.getProperty("erservicename") );}
```

ServiceSearch.searchForClosestToPerson()

The method searches for a service closest to a person.

Availability
IBM Security Identity Manager 7.0

Synopsis

ServiceSearch.searchForClosestToPerson(*person*, *profileName*)

Arguments

person
The `DirectoryObject` representing a person to use as the basis for the search.

profileName
Optional service profile name.

Returns
An array of `DirectoryObjects` representing the results of the search.

Description
This method searches for a service closest to a person.

Usage

```
//Search for AIX service closest to the person.
searchResult1 = ServiceSearch.searchForClosestToPerson(subject,
    "PosixAixProfile");

//Search for any service closest to the person.
searchResult2 = ServiceSearch.searchForClosestToPerson(subject);
```

UserAccess

The object extends the Account object and contains the data for a new account or changes to an existing account to provision the access, along with further information for the access.

Availability
IBM Security Identity Manager 7.0.

Inherits From
`Account`

Methods

getApprovalProcessID()
Returns the distinguished name of the approval process for gaining the access that was requested.

isNew()
Returns whether a new account is to be created by this access request, or an existing one is to be modified.

getAccessId()
Returns the identifier of the access that was requested.

getAccessName()
Returns the name of the access that was requested.

getAccessDescription()
Returns the description of the access that was requested.

getAccessOwner()
Returns the distinguished name of the owner of the access that was requested.

AccessRequestBatch

The object provides the `AccessRequestBatch` JavaScript object for use in the JavaScript environment of the access request batch notification templates.

The JavaScript object can be used in the ready-for-immediate-use notification templates - `Access Batch Processing Start Template` and `Access Batch Processing Complete Template`. The `AccessRequestBatch` object is a relevant data item in the `processAccessBatch` workflow definition. The workflow is not available for customization in workflow designer. To get the `AccessRequestBatch` JavaScript object, invoke `get()` on the relevant data ID `accessRequestBatch`.

Availability
IBM Security Identity Manager 6.0.0.17 and 7.0.1.6.

Provided by
`com.ibm.itim.workflow.script.WorkflowExtension`

Usage `var accessRequest = accessRequestBatch.get();`

Methods

getSubmittedAccessProvisioningList()
Returns the names of the accesses that were requested to be provisioned.

getSubmittedAccessDeprovisioningList()
Returns the names of the accesses that were requested to be deleted.

getSubmittedAccessUpdateList()
Returns the names of the accesses that were requested to be updated.

getAccessProvisioningStatusList()
Returns the access names and fulfillment status for the access provisioning requests that were made.

getAccessDeprovisioningStatusList()
Returns the access names and fulfillment status for the access deprovisioning requests that were made.

getAccessUpdateStatusList()
Returns the access names and fulfillment status for access update request made.

AccessRequestBatch.getSubmittedAccessProvisioningList()

This method returns the names of the accesses that were requested to be provisioned for a user. This method applies for access batch requests submitted from Identity Service Center only.

Availability
IBM Security Identity Manager 6.0.0.17 and 7.0.1.6.

Synopsis
`AccessRequestBatch.getSubmittedAccessProvisioningList()`

Returns
An array of access names that were requested to be provisioned for a user.

Description
This method returns the names of the accesses that were requested to be

provisioned for a user. This method applies for access batch requests submitted from Identity Service Center only.

Usage The following JavaScript is added in Access Batch Processing Start Template.

```
<RE key="Access Provisioning Request Submitted"/>:
<JS>
var result = '';
var accessList = accessRequestBatch.get().getSubmittedAccessProvisioningList();
 for (var i = 0; i &lt; accessList.length; i++)
 {
     var accessName = accessList[i];
     if (i == (accessList.length - 1)){
       result += accessName;
     }else{
       result += accessName + ', ';
     }
}
return result;
</JS>
```

AccessRequestBatch.getSubmittedAccessDeprovisioningList()

This method returns the names of the accesses that were requested to be deleted for a user. This method applies for access batch requests submitted from Identity Service Center only.

Availability
IBM Security Identity Manager 6.0.0.17 and 7.0.1.6.

Synopsis
`AccessRequestBatch.getSubmittedAccessDeprovisioningList()`

Returns
An array of access names that were requested to be deleted for a user.

Description
This method returns the names of the accesses that were requested to be deleted for a user. This method applies for access batch requests submitted from Identity Service Center only.

Usage The following JavaScript is added in Access Batch Processing Start Template.

```
<JS>
var result = '';
var accessList = accessRequestBatch.get().getSubmittedAccessDeprovisioningList();
 for (var i = 0; i &lt; accessList.length; i++)
 {
     var accessName = accessList[i];
     if (i == (accessList.length - 1)){
       result += accessName;
     }else{
       result += accessName + ', ';
     }
}
return result;
</JS>
```

AccessRequestBatch.getSubmittedAccessUpdateList()

This method returns the names of the accesses that were requested to be updated for a user. This method applies for access batch requests submitted from Identity Service Center only.

Availability

IBM Security Identity Manager 6.0.0.17 and 7.0.1.6.

Synopsis

`AccessRequestBatch.getSubmittedAccessUpdateList()`

Returns

An array of access names that were requested to be updated for a user.

Description

This method returns the names of the accesses that were requested to be updated for a user. This method applies for access batch requests submitted from Identity Service Center only.

Usage The following JavaScript is added in Access Batch Processing Start Template.

```
<JS>
var result = '';
var accessList = accessRequestBatch.get().getSubmittedAccessUpdateList();
 for (var i = 0; i &lt; accessList.length; i++)
 {
     var accessName = accessList[i];
     if (i == (accessList.length - 1)){
       result += accessName;
     }else{
       result += accessName + ', ';
     }
}
return result;
</JS>
```

AccessRequestBatch.getAccessProvisioningStatusList()

This method returns an array of access names and the fulfillment status for access provisioning requests. The status can be `Fulfilled`, `Not Fulfilled`, `Pending`, or `Unknown`.

Availability

IBM Security Identity Manager 6.0.0.17 and 7.0.1.6.

Synopsis

`AccessRequestBatch.getAccessProvisioningStatusList(String processId)`

Arguments

The process ID of the access batch request process.

Returns

An array of access names and their statuses in the following format.

access name-access status

Description

This method returns an array of access names and the fulfillment status for access provisioning requests. The status can be `Fulfilled`, `Not Fulfilled`, `Pending`, or `Unknown`.

Usage The following JavaScript is added in Access Batch Processing Complete Template.

```
</JS>
<RE key="Access Provisioning Status"/>: <JS>
 var result = '';
 var accessStatusList = accessRequestBatch.get().getAccessProvisioningStatusList(process.id);
 for (var i = 0; i &lt; accessStatusList.length; i++)
 {
     var accessNameStatus = accessStatusList[i];
     if (i == (accessStatusList.length - 1)){
       result += accessNameStatus;
```

```
    }else{
      result += accessNameStatus + ', \n        ';
    }
 }
return result;
</JS>
```

AccessRequestBatch.getAccessDeprovisioningStatusList()

This method returns an array of access names and the fulfillment status for access removal requests. The status can be `Fulfilled`, `Not Fulfilled`, `Pending`, or `Unknown`.

Availability

IBM Security Identity Manager 6.0.0.17 and 7.0.1.6.

Synopsis

`AccessRequestBatch.getAccessDeprovisioningStatusList(String processId)`

Arguments

The process ID of the access batch request process.

Returns

An array of access names and their statuses in the following format.

access name-access status

Description

This method returns an array of access names and the fulfillment status for access removal requests. The status can be `Fulfilled`, `Not Fulfilled`, `Pending`, or `Unknown`.

Usage The following JavaScript is added in Access Batch Processing Complete Template.

```
<JS>
 var result = '';
 var accessStatusList = accessRequestBatch.get().getAccessDeprovisioningStatusList(process.id);
 for (var i = 0; i &lt; accessStatusList.length; i++)
 {
     var accessNameStatus = accessStatusList[i];
     if (i == (accessStatusList.length - 1)){
       result += accessNameStatus;
     }else{
       result += accessNameStatus + ', \n        ';
     }
 }
return result;
</JS>
```

AccessRequestBatch.getAccessUpdateStatusList()

This method returns an array of access names and the fulfillment status for access update requests. The status can be `Fulfilled`, `Not Fulfilled`, `Pending`, or `Unknown`.

Availability

IBM Security Identity Manager 6.0.0.17 and 7.0.1.6.

Synopsis

`AccessRequestBatch.getAccessUpdateStatusList(String processId)`

Arguments

The process ID of the access batch request process.

Returns

An array of access names and their statuses in the following format.

access name-access status

Description

This method returns an array of access names and the fulfillment status for access update requests. The status can be `Fulfilled`, `Not Fulfilled`, `Pending`, or `Unknown`.

Usage The following JavaScript is added in Access Batch Processing Complete Template.

```
<JS>
 var result = '';
 var accessStatusList = accessRequestBatch.get().getAccessUpdateStatusList(process.id);
 for (var i = 0; i &lt; accessStatusList.length; i++)
 {
     var accessNameStatus = accessStatusList[i];
     if (i == (accessStatusList.length - 1)){
       result += accessNameStatus;
     }else{
       result += accessNameStatus + ', \n        ';
     }
 }
return result;
</JS>
```

Chapter 11. Provisioning policy parameter usage scenarios

The following scenarios illustrate usage of provisioning policy parameters.

Scenario 1: No attributes defined

If no parameter values are selected for a multi-valued attribute, all values are valid for this attribute.

If a parameter is added for a multi-valued attribute with the parameter type as `Allowed` (valid), all other values for this attribute are implicitly excluded for this policy.

If an attribute value is added to a policy as valid, all other values are implicitly excluded for that parameter for the policy.

For multiple applicable entitlements, a valid attribute value is determined by the join directive for the attribute. See the following scenarios.

Scenario 2: Priority-based provisioning policy join directive

The following table identifies two examples of provisioning policies:

Table 10. Provisioning policy examples

Policy	Description
Policy 1	Priority = 1 Attribute: `erdivision` = divisionA, `enforcement` = DEFAULT
Policy 2	Priority = 2 Attribute: `erdivision` = divisionB, `enforcement` = MANDATORY

Because Policy 1 has a higher priority, only Policy 1's definition for the `erdivision` attribute is used. Policy 2's definition for the `erdivision` attribute is ignored.

During policy validation, including reconciliation with policy check option turned on, `divisionA` might exist on the `erdivision` attribute. All other values are valid, because the only policy that is being considered in a priority join is Policy 1, which has DEFAULT enforcement on `erdivision`. DEFAULT enforcement by itself is interpreted as valid for all values, but the default value is the value specified on the new account.

Note: A priority join directive is the default join directive for all single-valued and string-typed attributes.

Scenario 3: Union-based provisioning policy join directive

The following table identifies two example provisioning policies:

Table 11. Sample provisioning policies

Policy	Description
Policy 1	Priority = 1 Attribute: `localgroup` = groupA, `enforcement` = DEFAULT
Policy 2	Priority = 2 Attribute: `localgroup` = groupB, `enforcement` = MANDATORY

Because the join directive is defined as UNION, the resulting policy uses the following definitions for the policies:

- During account creation, `localgroup` is defined as `groupA` and `groupB`.
- During reconciliations, `localgroup` is defined as `groupB` if the attribute is undefined or incorrectly defined.

Note: A union join directive is the default join directive for multi-valued attributes.

Chapter 12. Provisioning policy entitlement parameters

Provisioning policy parameters help system administrators define the attribute values that are required and the values that are allowed.

The following parameter types are valid:

- Constant value
- Null
- JavaScript
- Regular expression

The provisioning parameters in an entitlement can be statically or dynamically defined. Parameters are defined statically by selecting the constant parameter type and specifying literal values, such as strings or integers. For example, an attribute can be defined as Domain Users or Power® Users. A dynamically defined parameter value can be based on a JavaScript function. A range of values can be defined that use a regular expression.

Parameters can also be specified as `Null`, indicating that the parameter does not have a value. This situation is equivalent to having a JavaScript parameter type with a value of `return null;`

Provisioning parameters for single-valued attributes can be based only on JavaScript code or a constant. The provisioning parameters of a multi-valued attribute can use a constant, JavaScript code, or regular expression for their values.

However, a regular expression can be used only for provisioning parameter enforcement of the Allowed or Excluded type.

Provisioning policy constant

A static, constant value can be assigned to an entitlement parameter for a single or multivalued attribute with the provisioning policy Constant parameter type. You can define a provisioning parameter with a literal static value. You can enter the value or select a value based on the field widget.

Provisioning policy Null types

The null parameter type can be used to specify a null value for an account attribute. If the value of a parameter is specified as null with mandatory enforcement, no value is valid for that attribute. You can specifically define null value for the provisioning parameter, which is equivalent to specifying empty for the value.

Provisioning policy JavaScript functions

You can use a script to define provisioning parameters.

The provisioning parameters of an entitlement within a provisioning policy can be defined by a script. The context of the script is

- The person for whom the entitlement is being enforced.

- The service the entitlement is protecting.
- The `eruid` attribute of the target account.

The context of the script includes the following elements:

Subject
Owner of the account.

Service
Service on which the account exists or to be created.

uid User ID of the account.

Context
Information about the parameter evaluation, which can be validation of a new account or validation of existing account.

A special object named *parameters* is available for `eruid` to evaluate the script in the context of provisioning policy parameters. To obtain its value, use the following syntax:

```
parameters.eruid[0]
```

The value of zero in this syntax returns the first value of the array object.

A JavaScript object named *subject* represents a user for whom the entitlement is being enforced. The service is represented by another JavaScript data model entity named *service*. The script author uses both the subject and service object to access attributes of these objects.

The values of attributes of objects that are part of the evaluation context can also be retrieved with the IBM Security Identity Manager custom JavaScript functions.

To use JavaScript to define the value of an attribute, the JavaScript parameter type must be selected. Select **JavaScript/Constant** in the **Expression Type** field.

The following examples demonstrate the use of IBM Security Identity Manager custom JavaScript functions within provisioning policies. For a complete reference to all custom JavaScript functions, see the JavaScript Extension Reference.

Person attributes

Syntax:

```
subject.getProperty(String rowAttrName)
```

Example:

```
subject.getProperty("sn")[0];
```

Example:

```
# Concatenates user's given name and family name with space in between.
# Resulting string value may be used to on account attribute such as
# Description.
{subject.getProperty("givenname")[0] + " " + subject.getProperty("sn")[0];}
```

Example:

```
# Set a user's Password attribute to the user's Shared Secret Attribute
# (if the account is automatically provisioned)

{
```

```
   function passInit()
     {var password = subject.getProperty("ersharedsecret");
     if (password.length > 0){
         return password[0];
     } else {
         return ""
     }
   }return
   passInit();
}
```

Search for person

Syntax:

```
PersonSearch.searchByFilter(String profileName, String filter, [int scope])
```

where scope =1 is a single level search and scope =2 is a subtree search.

Example:

```
PersonSearch.searchByFilter("Person", "(sn=Smith)", 1);
```

Search for service

Syntax:

```
ServiceSearch.searchByFilter(String filter, [int scope])
```

where scope=1 is a single level search and scope=2 is a subtree search.

Example:

```
ServiceSearch.searchByFilter("(erntlocalservername=*srv)", 1);
```

Service closest to the person

Syntax:

```
ServiceSearch.searchForClosestToPerson(Person person, [int scope])
```

where scope=1 is a single level search and scope=2 is a subtree search.

Example:

```
ServiceSearch.searchForClosestToPerson(subject);
```

Name of the business unit in which the person is located

Syntax:

```
subject.getProperty(String propertyName)
```

Example:

```
subject.getProperty("Parent")[0].name;
```

Specifying the current account Uid

Syntax:

```
uid = parameters.eruid[0];
```

Example:

```
var accountId = parameters.eruid[0];
```

Enrole.toGeneralizedTime statement

Syntax:

```
Enrole.toGeneralizedTime(Date date)
```

Examples:

Using the function to return today's date string:

```
var gt = Enrole.toGeneralizedTime(new Date());
```

Using the function to return today's date string as a default attribute:

```
{Enrole.toGeneralizedTime(new Date())}
```

Enrole.toMilliseconds statement

Syntax:

```
Enrole.toMilliseconds(String generalizedTime)
```

Examples:

```
var millis = Enrole.toMilliseconds("200101012004Z");
var date = new Date(millis);
```

Provisioning policy regular expressions

Regular expressions are used to define a matching pattern that is checked against given text. Within IBM Security Identity Manager, regular expressions define allowed and excluded parameter values.

Within IBM Security Identity Manager, regular expressions define allowed and excluded parameter values. Parameter values with regular expressions are used against existing attribute values to determine which ones are valid.

To use a regular expression for a provisioning parameter value, select **Regular Expression** in the Expression Type menu.

Note: Regular Expression can be used only with excluded or allowed attributes. See the `regexp` Java library for a syntax reference.

Chapter 13. Service selection policy JavaScript

A service selection policy identifies the service type for the service returned, and the JavaScript specifies the service. For example, the service definition can be based on attributes of an account owner.

Service selection policy JavaScript objects

The service selection policy JavaScript returns an object that represents a IBM Security Identity Manager service entity.

The "subject" JavaScript object is a Person object that represents the account owner. Attributes of the Person can be used to construct filters to search and return the service. The `ServiceModelExtension` is available for Service Selection policy by default.

The following list includes APIs for the `ServiceSearch` JavaScript object that can be used to return the service:

- `ServiceSearch.searchByName`
- `ServiceSearch.searchByFilter`
- `ServiceSearth.searchForClosestToPerson`

See a JavaScript API reference guide for detailed information for these APIs.

Service selection policy script example

This section includes examples of Service Selection policy scripts.

Service selection based on family name

The following script returns a service instance based on the family name of the account owner. Other person attributes such as job title and location can be used to select service, as in this example.

```
function selectService() {
  var sn = subject.getProperty("sn")[0];
  var serviceInstance = null;
  if(sn=="Jones") {
    serviceInstanceArr = ServiceSearch.searchByFilter(
      "(erservicename=NT40X)", 1);

    if (serviceInstanceArr != null && serviceInstanceArr.length > 0)
      serviceInstance = serviceInstanceArr[0];
  } else {
    serviceInstanceArr = ServiceSearch.searchByFilter(
      "(erservicename=NT40Y)", 1);

    if (serviceInstanceArr != null && serviceInstanceArr.length > 0)
      serviceInstance = serviceInstanceArr[0];
  }
  return serviceInstance;
}
return selectService();
```

Searching for the closest service to the person

The following example searches for the service closest to the level of the person, based on the organization tree structure.

```
function selectService() {
  var services = ServiceSearch.searchForClosestToPerson(subject);

  if (services != null && services.length > 0) {
     return services[0];
  }
}
return selectService();
```

Chapter 14. SubForm control type

The SubForm control type provides a means to use custom user interfaces for complex multi-valued attributes.

This control type is used infrequently by some IBM Security Identity Manager adapters.

SubForm is a special control type used to start a servlet, JSP, or static HTML page from a window that opens from a custom IBM Security Identity Manager form. Use subforms to submit an arbitrary number of parameter names and values to a custom servlet or JSP. They are used to create custom user interfaces for complex multi-valued attributes.

Table 12. SubForm parameters

Parameter	Description	Value
`customServletURI`	The URI to the servlet, JSP, or static HTML page to be started from the main form. If a servlet is implemented and deployed in the default web application for IBM Security Identity Manager, the value for this parameter is the same as the *`URL-pattern`* value defined by`web.xml` in the *`servlet-mapping`* tag, without the slash (/). If a JSP is implemented, the value for this parameter is the JSP file name that includes the jsp file extension. This parameter is required on all subforms.	Servlet name or JSP file name such as `sample.jsp`
`Parameter Name`	Arbitrary parameter name and value that is included in the HTTP request that starts the resource at `customServletURI`. For example: `objectClass=erracfgrp`	*`Parameter Value`*

From the Appliance Dashboard on the IBM Security Identity Manager virtual appliance console, click **Configure** > **Advanced Configuration** > **Custom File Management**. From the **All Files** tab, go to `directories/utilities` and download the `extensions.zip` file and extract it. Go to `\extensions\7.0\examples\subform`.

SubForm contextual parameters

As a child element of a main form, a SubForm is passed contextual parameters that help identify the context from which it is started.

These contextual parameters are included in the HTTP Request that brings up the SubForm:

Table 13. SubForm parameters

HTTP (contextual) Parameter Name	Person Create	Person Modify	Account Create	Account Modify
`target_dn`	empty	DN of Person	DN of account owner	DN of the account
`container_dn`	DN of the organization tree container where the Person is created.	DN of the organization tree container where the person is located.	DN of account owner	DN of the account owner
`search_base`	empty	empty	DN of service	DN of the service instance on which an account is provisioned

To assign the `target_dn` HTTP parameter value to a String declared inside a servlet:

```
String targetDN = request.getParameter("target_dn");
```

Account Modify example

For example, for Account Modify, the value of contextual parameters are:

target_dn
Is the DN of the entity whose attributes are displayed on the main form. If the SubForm is placed on a RACF® account form, this parameter value is the `DistinguishedName` of the RACF account.

container_dn
Is the entity container or parent. For example, if the SubForm is placed on a Person form, this parameter value is the `DistinguishedName` of the parent or container of the person. The container can be an organization, organizational unit, admin domain, or location.

search_base
For example, if the SubForm is placed on a RACF account form, this parameter value is the `DistinguishedName` for the RACF service instance on which the account is provisioned.

HTTP request parameter naming convention

A naming convention used on SubForm parameters prevents collisions with other parameters (such as contextual parameters).

The naming convention for SubForm parameters is:

```
[prefix].[attributename].[parametername]
```

where:

prefix `property.data`

attributename
Name of the attribute on which the SubForm is placed on the main form.

parametername

Name used in the SubForm Editor dialog. For example, an HTTP parameter named `property.erracfconnectgroup.objectClass` would contain the value defined in the SubForm editor dialog assigned to `objectClass`.

To assign this value to a string declared inside a servlet:

```
String objectClass =
request.getParameter("property.data.erracfconnectgroup.objectClass");
```

Process to write a SubForm

To write a custom SubForm, create a servlet that generates the HTML user interface to manage the value of the attribute.

To save the value, create an instance of `com.ibm.itim.common.AttributeValue` and bind it to a user's `HttpSession` with the key defined in `com.ibm.itim.webclient.util.FormData` (on one line):

```
AttributeValue av = new AttributeValue("attributename", "customValue");
 HttpSession session = request.getSession(false);
  session.setAttribute("subFormAttrValue", av);
```

This ensures that the value gets picked up and added to the form data collected from the fields when the main form is submitted.

Chapter 15. Supplemental property files

The following section provides detailed information about the property keys and values contained in the IBM Security Identity Manager supplemental property files.

Properties files

Java properties files define attributes that allow customizing and control of the Java software.

Standard system properties files and custom properties files are used to configure user preferences and user customization. A Java properties file defines the values of named resources that can specify program options such as database access information, environment settings, and special features and functions.

A properties file defines named resources with a property key and value pair format:

```
property-key-name=value
```

The *property-key-name* is an identifier for the resource. The *value* is usually the name of the actual Java object that provides the resource, or a String representing the value of the property key, such as `database.name=itimdb`. The statement syntax allows spaces before and after the equal (=) sign, and can span multiple lines if you place a line continuation character \ (a backslash) at the end of the line. For more information about statement syntax, see Java language references.

IBM Security Identity Manager uses a number of properties files to control the program and to allow user customization of special features.

Modifiable property files

This table lists the IBM Security Identity Manager properties files that you can modify.

Table 14 lists the IBM Security Identity Manager properties files. Most files are in the **Identity server property files** tab of the Update Property page from the **Appliance Dashboard** of the IBM Security Identity Manager virtual appliance console.

Additional properties files are not configurable. Do not modify them.

Table 14. Properties files

Property file name	Description
`adhocreporting`	The `adhocreporting.properties` file supports the custom reporting module.
`CustomLabels`	The property key and value pairs in the `CustomLabels.properties` file are used by the Security Identity Manager user interface to display the label text for forms.
`DataBaseFunctions.conf`	The custom reporting feature of Security Identity Manager allows you to use database functions when designing custom report templates.

Table 14. Properties files (continued)

Property file name	Description
enRole	The enRole.properties system configuration file contains many of the properties that are used to configure IBM Security Identity Manager.
enroleAuditing	The property key and value pairs in the enroleAuditing.properties file are used to enable or disable the tracking of changes made by a Security Identity Manager user to business objects such as person, location, service, and other objects, or configuration of the system.
enRoleAuthentication	The enRoleAuthentication.properties file specifies the type of method that is used by the Security Identity Manager Server to authenticate users and identifies the Java object that provides the specified authentication mechanism.
enRoleDatabase	The enRoleDatabase.properties file specifies attributes that support the relational database used by Security Identity Manager.
enRoleLDAPConnection	The enRoleLDAPConnections.properties file provides standard configuration settings that allow successful communication between Security Identity Manager and the LDAP directory server.
enRoleLogging	The enRoleLogging.properties file specifies attributes that govern the operation of the **jlog** logging and tracing API that is bundled with Security Identity Manager.
enrolepolicies	The enrolepolicies.properties file provides standard and custom settings that support the functions of the provisioning policy.
enroleStartup	
enroleworkflow	The enroleworkflow.properties file specifies the XML file mappings for system-defined workflows.
fesiextensions	The fesiextensions.properties file (deprecated) provides support for Free EcmaScript Interpreter (FESI) JavaScript extensions before Version 5.0 of Security Identity Manager. Do not author *new* extensions using this deprecated architecture.
helpmappings	The helpmappings.properties file allows a customer to replace the installed Security Identity Manager help system with an alternative help system.
reportingLabels	This properties file is like other labels-related properties files such as labels.properties, or customLabels.properties, and holds labels that are used by Reports.
reporttabledeny	By default, this property holds a list of Security Identity Manager tables that are used by various Security Identity Manager components to store internal or configuration data that is inappropriate for a report.
scriptframework	For *all* new JavaScript extensions, use the scriptframework.properties file to configure script extensions and other scripting functions.
SelfServiceHelp	The SelfServiceHelp.properties file can be used to redirect help to a custom location for customers who want to have their own help content for the self-service user interface.

Table 14. Properties files (continued)

Property file name	Description
SelfServiceHomePage	The SelfServiceHomePage.properties file is used to configure the sections of the initially installed home page for the self-service user interface. You can add or remove tasks, and update icon URLs and labels of the home page from this file.
SelfServiceUI	The SelfServiceUI.properties file controls miscellaneous properties of the self-service user interface.
ui	The ui.properties file specifies attributes that affect the operation and display of the Security Identity Manager graphical user interface.

Modifying values of the property files from the IBM Security Identity Manager virtual appliance console

1. From the top-level menu of the **Appliance Dashboard**, select **Configure** > **Advanced Configuration** > **Update Property** to display the Update Property page. For more information, see Managing the server properties.
2. In the Update Property page, click the **Identity server property files** tab.
3. Select a properties file from its list. Depending on any property names and its values that are associated with the selected file, the right pane displays all of them.
4. Select a property name.

   Note: You can also use the search box to find a specific property name that you want to update.
5. Click **Edit** to open the Update property window.
6. Edit the value in the **Property value** field.
7. Click **Save Configuration**.
8. On the **Appliance Dashboard**, you might be asked to restart the IBM Security Identity Manager Server.

adhocreporting.properties

The adhocreporting.properties file supports the custom reporting module.

Table 15 defines the properties used to configure reporting.

Table 15. adhocreporting.properties properties

Report Generation	
reportPageSize	
	Indicates the number of rows that are displayed on each page of a PDF report. The maximum number of rows on a page must not exceed 45. Example (default): reportPageSize=45
reportColWidth	

Table 15. adhocreporting.properties properties (continued)

	Indicates the width, in centimeters (cm), of the report column in a PDF report output. You can adjust the size of all columns by modifying this value. **Note**: 2.54 cm equals 1 inch. Example (default): `reportColWidth=20`
Access Control Item Enforcement on Report Data Generated	
`availableForNonAdministrators`	
	Specifies whether to synchronize access control item-related information during data synchronization. Set this value to `true` to enable non-administrators to run reports. Set this value to `false` to disable all functions related to non-administrator execution of reports, such as access control item data synchronization and setting report access control items on reports. Example: `availableForNonAdministrators=true`
Incremental schema Enforcement using Incremental Data Synchronizer	
`enableDeltaSchemaEnforcer`	
	Specifies whether to synchronize any schema changes in reporting. Schema changes include new mappings that were created or existing mappings that were removed with the Schema Designer. When set to `true`, the Incremental Data Synchronizer manages the attributes that are mapped (changed) in the Schema Designer since the last full data synchronization was run. When set to `false`, the Incremental Data Synchronizer does not synchronize the attributes which are mapped (changed) since the last full data synchronization was run. Example (default): `enableDeltaSchemaEnforcer=false`
Data Synchronization	
`changelogEnabled`	
	Specifies whether the Incremental Data Synchronizer is used. Values include: • **true** – Incremental Data Synchronizer is configured • **false** – Incremental Data Synchronizer is not configured Example (default): `changelogEnabled=false`

Table 15. adhocreporting.properties properties (continued)

`changelogBaseDN`	
	Specifies the DN in the directory where the change log is configured. Example (default): `changelogBaseDN=cn=changelog`
`changeLogFetchSize`	
	Specifies the number of change logs to be fetched at one time from the directory server. A value of 0, or a negative value, results in no fetch restriction. Fetch restriction is useful when the directory server cannot be heavily loaded for a time. For example, retrieving 100,000 change log entries at a time can delay the directory server response time for a few minutes. Example (default): `changeLogFetchSize=200`
`maximumChangeLogsToSynchronize`	
	Specifies the maximum number of change logs to be synchronized in a single use of the Incremental Data Synchronizer. Consider the available system memory and CPU utilization that is required for other processes in the system when you set this property. If the value is set to zero or a negative value, the Incremental Data Synchronizer synchronizes all change log entries. Example (default): `maximumChangeLogsToSynchronize=10000`
`changeLogsToAnalyzeBeforeSynchronization`	

	Specifies the number of fetched change log entries to be analyzed before all analyzed entries are synchronized to the database.
	
	For example, consider the following values:
	`changeLogFetchSize=500`
	`changeLogsToAnalyzeBeforeSynchronization=20000`
	`maximumChangeLogsToSynchronize=100000`
	
	500 change log entries are considered one batch. After 20,000 change log entries (40 batches) are analyzed, data synchronization occurs. This process repeats until 100,000 entries are analyzed (5 synchronizations).
	
	Setting this value to 0 or a negative value results in synchronization of all fetched change log entries.
	
	Example (default):
	`changeLogsToAnalyzeBeforeSynchronization=5000`

`enableChangelogPruning`	

Table 15. adhocreporting.properties properties (continued)

	Specifies whether `changelog` entries need to be pruned after they are successfully synchronized. This property takes effect only for the SunOne Version 5.2 directory server. For the IBM Security Directory Server, see its documentation about pruning `changelog` entries. Example (default): `enableChangelogPruning=false`
`itimAdminID`	
	Specifies the administrator ID required to run the Incremental Data Synchronizer in a z/OS® environment. For example: `itimAdminID=myadminid`
`itimAdminCredential`	
	Specifies the Security Identity Manager password required to run the Incremental Data Synchronizer in a z/OS environment. For example: `itimAdminCredential=myadmincredential`
`createIndex`	
	Specifies whether to create database indexes for frequently used database columns. If this property is set to `true`, reports are generated more quickly. Valid values for this property are: • **true** – Creates indexes for columns that are used by reporting. Enabling this value might increase the data synchronization time. • **false** – Does not create indexes during data synchronization. Disabling this value might increase the time that is needed to generate reports. Example (default): `createIndex=true`
`reportIndexes`	

Table 15. adhocreporting.properties properties (continued)

	Specifies a set of a set of <ENTITY:(ATTR1 ORDER1, ATTR2 ORDER2, ...)> values on which indexes are created.
	
	Both single and compound indexes can be created with this property. If you are creating a single index, use the name of entity that you see in the report designer or schema mapping.
	
	If you are defining a compound index, specify the exact table name, such as Account or Person_cn, instead of the entity name. You can specify an optional order asc or desc for an index. Observe the usage of a semi-colon as the delimiter between indexes. You must maintain the syntax of this property correctly, or indexes might not get created successfully.
	
	If you add additional indexes, follow the syntax for these default indexes:
	`reportIndexes=Person:cn asc;Account:eraccountcompliance;`
	`Account:(eraccountstatus asc);Account:erlastaccessdate asc;`
	`Account:eruid asc;Service:(servicetype asc);`
	`Service:erservicename asc;ProvisioningPolicy:erpolicyitemname asc;`
	`ProvisioningPolicy:erpolicytarget asc;`
	`ProvisioningPolicy:erpolicymembership asc;Role:errolename asc;`
	`Account:(eraccountstatus asc, erservice asc);`
	`Person_cn:(dn, cn);Account_owner:(dn asc, owner asc)`

`sqlBatchSize`	
	Indicates the size of batch updates that are processed during data synchronization. To improve performance, set this value to a larger number. This value is affected by the specific database settings for the transaction log file size, a database property. Setting the value too high might cause data synchronization to fail. Always use the default value of 50 to avoid data synchronization failure. A value of 0, or a negative value, causes all SQL updates to be processed as a single batch. Example (default): `sqlBatchSize=50`
`attribsSkippedInSchema`	
	These attributes contain XMLs as data. The reporting engine currently does not support reporting on these attributes. Example (on one line): `attribsSkippedInSchema=erEntitlements erAcl erHistoricalPassword erJavascript erLostPasswordAnswer erPassword erPlacementRule erxforms erXML`
`reportsAllowedAttributes`	
	A set of attributes on which reporting engine does not enforce attribute-level access control. Example (default): `reportsAllowedAttributes=servicetype`
`reportsAllowedEntities`	

Table 15. adhocreporting.properties properties (continued)

	A set of entities on which reporting engine does not enforce attribute-level access control. Example (default): `reportsAllowedEntities=RecertificationPolicy,Group`
`reservedWords`	
	Database reserved words. These words are not used as table/column names during Schema Mapping and Data Synchronization. Example (on one line): `reservedWords=ALL ADD ALTER BACKUP BEGIN BY BULK CASCADE CHECK CHECKPOINT CLUSTORED COLUMN CREATE CURRENT DUMMY DOMAIN DELETE DEFAULT DISTINCT DROP FORIGN FROM GROUP IDENTITY IDENTITY_INSERT IDENTITYCOL INSERT IN LIKE SET SELECT TABLE VALUES ORDER UID WHERE`
`disallowedChars`	
	Characters that are not part of Table/Column name in database. If the entity/attribute name contains one or more of these characters, the characters are removed from the table or column name. In the following example, the double backslashes (\\) are used as escape characters. Example (default): `disallowedChars=~!@#%^&*()+{}|:\"<>? -=[]\\;',./`
`disallowedCharsForStart`	
	Characters are not used as the starting character of table or column name. In the following example, the double backslashes (\\) are used as escape characters. Example (default): `disallowedCharsForStart=~!@#$%^&*()_+{}|:\"<>? -=[]\\;',./0123456789`
`maxTableNameLength`	
	Default maximum length for a table name. Example (default): `maxTableNameLength=30`
`maxColumnNameLength`	
	Default maximum length for a column name. Example (default): `maxColumnNameLength=30`
`maxTableNameLength_DB2`	
	Maximum name length for a table name in DB2®. Example (default): `maxTableNameLength_DB2=128`
`maxColumnNameLength_DB2`	

Table 15. adhocreporting.properties properties (continued)

	Maximum name length for a column name in DB2. Example (default): `maxColumnNameLength_DB2=30`
`maxTableNameLength_ZDB2`	
	Maximum name length for a table name in DB2 z/OS. Example (default): `maxTableNameLength_ZDB2=128`
`maxColumnNameLength_ZDB2`	
	Maximum name length for a column name in DB2 z/OS. Example (default): `maxColumnNameLength_ZDB2=30`
`maxTableNameLength_ORACLE=30`	
	Maximum name length for a table name in Oracle. Example (default): `maxTableNameLength_ORACLE=30`
`maxColumnNameLength_ORACLE`	
	Maximum name length for a column name in Oracle. Example (default): `maxColumnNameLength_ORACLE=30`
`maxTableNameLength_MS_SQL_SERVER`	
	Maximum name length for a table name in Microsoft SQL Server. Example (default): `maxTableNameLength_MS_SQL_SERVER=128`
`maxColumnNameLength_MS_SQL_SERVER`	
	Maximum name length for a column name in Microsoft SQL Server. Example (default): `maxColumnNameLength_MS_SQL_SERVER=128`
`populateGroupMembers`	

Table 15. adhocreporting.properties properties (continued)

	Specifies whether Service group membership changes need to be synchronized during incremental synchronization. Service group membership information is stored in the GROUPMEMBERS table. Valid values for this property are: • **true** – Synchronizes membership changes to service groups and accesses (for example, because of a new access request). • **false** – Does not synchronize group membership changes, since this type of synchronization is performance intensive. Example (default): `populateGroupMembers=false`

CustomLabels.properties

The property key and value pairs in the `CustomLabels.properties` file are used by the IBM Security Identity Manager user interface to display the label text for forms.

The key name must be entirely lowercase in each property key and value pair.

A separate `CustomLabels.properties` file exists for each individual language supported by IBM Security Identity Manager.

This file is used to provide localized versions of graphical user interface elements when IBM Security Identity Manager is installed in international environments.

Add the property key and value pairs in the `CustomLabels.properties` properties file to display any labels.

For example, to display a two word access type - **Business Applications**,

1. Specify the access key as **businessApplications**. The access type key cannot contain a space.
2. Specify the value as **Business Applications**.

The entry in the `CustomLabels.properties` file to have "Business Applications" displayed in the user interface as the access type is **businessApplications=Business Applications**.

Access types that are part of a hierarchy of types have a special notation that you must use in the CustomLabels.properties file. Each node of the hierarchy must be in the key and separated by a period (.). For example, an access type that is called **Applications** has a child **businessApplications**. You want **businessApplications** to display as "Business Applications". The entry that you define in the `CustomLabels.properties` file is **Applications.businessApplications=Business Applications**.

A file name extension identifies the specific language. For example:

English

```
CustomLabels_en.properties
```

Japanese

```
CustomLabels_ja.properties
```

enroleAuditing.properties

The property key and value pairs in the `enroleAuditing.properties` file are used to enable or disable the tracking of changes made by a Security Identity Manager user to business objects such as person, location, service, and other objects, or configuration of the system.

Any user request to change the IBM Security Identity Manager directory store or database can be audited and published in a report.

The following is a comprehensive list of events audited:

- ACI Management (**Add**, **Add Authorization Owner**, **Delete**, **Delete Authorization Owner**, **Modify**)
- Account Management (**Add**, **Adopt**, **Change Password**, **Delete**, **Modify**, **Orphan**, **Password Pickup**, **Restore**, **Suspend**, **Synchronize Password**)
- Access Management (**Add**, **Remove**)
- Access Configuration (**Add**, **Remove**, **Modify**)
- Authentication (**Authenticate ITIM user**)
- Container Management (**Add**, **Delete**, **Modify**)
- Delegate Authority (**Add**, **Delete**, **Modify**)
- Entitlement Workflow Management (**Add**, **Delete**, **Modify**)
- Entity Operation Management (**Add**, **Delete**, **Modify**)
- IBM Security Identity Manager Configuration (**Add**, **Delete**, **Enforce**, **Install Profile**, **Modify**, **Uninstall Profile**)
- Group Management (**Add**, **Add Member**, **Delete**, **Modify**, **Remove Member**)
- Migration (**Agent Profile Install**, **Start Export**, **Start Import**, **Stop Export**, **Stop Import**)
- Role Management (**Add**, **Add Member**, **Delete**, **Modify**, **Remove Member**)
- Person Management (**Add**, **Delete**, **Modify**, **Restore**, **Self Register**, **Suspend**, **Transfer**)
- Policy Management (**Add**, **Commit Draft**, **Delete**, **Enforce Entire Policy**, **Modify**, **Save as Draft**, **Add Account Template**, **Change Account Template**, **Remove Account Template**)
- Reconciliation (**Run Recon**, **Set Recon Unit**, **Set Service Recon Parameters**)
- Runtime Events (**Start IBM Security Identity Manager**, **Stop IBM Security Identity Manager**)
- Self Password Change (**Change Password**, **Reset Password**)
- Service Management (**Add**, **Add Adoption Rule**, **Delete**, **Delete Adoption Rule**, **Modify**, **ModifyAdoption Rule**)
- Service Policy Enforcement (**Correct Non-Compliant**, **Mark Non-Compliant**, **Suspend Non-Compliant**, **Use Global Setting**, **Use Workflow For Non-Compliant**)

Audited information specifically includes:

- The identity of the user who takes the action.
- The time the action was taken.
- The type of action taken.
- The data effected by the action.

Table 16 defines the properties used to configure how the auditing feature behaves.

Table 16. enroleAuditing.properties properties

IBM Security Identity Manager audit configuration settings	
`itim.auditing`	
	Specifies whether to enable or disable auditing for IBM Security Identity Manager events. Valid values include: • **true** – IBM Security Identity Manager events are audited • **false** – IBM Security Identity Manager events are not audited, regardless of the settings of individual events or categories Example (default): `itim.auditing=true`
`itim.auditing.retrycount`	
	The number of times auditing is tried again in case of failure. Valid values include any integer. Example (default): `itim.auditing.retrycount=1`
`itim.auditing.retrydelay`	
	The wait time in milliseconds before trying again. Example (default): `itim.auditing.retrydelay=5000`
`enrole.auditing.errorpopup.enabled`	
	Enables or disables the process failure display. Example (default): `enrole.auditing.errorpopup.enabled=false`
`enrole.auditing.errorpopup.fields`	

	The process failure display always contains these attributes and their values:
	`{name, subject, type, result_summary}`
	You can additionally specify one or more of these attributes:
	`{subject, comments, name, type, requester_type, requester_name, description, scheduled, started, completed, lastmodified, submitted, state, notify, requestee_name, subject_profile, subject_service, result_summary, result_detail}`
	Example:
	`enrole.auditing.errorpopup.fields=subject, comments`

`enrole.auditing.errorpopup.textwrap`	

Table 16. enroleAuditing.properties properties (continued)

	Specifies whether the text wraps at the end of the display. Example (default): `enrole.auditing.errorpopup.textwrap=false`
`enrole.auditing.pageSize`	
	Specifies the page size in lines that displaying unsuccessful processes or activities on the failed activity popup. Example (default): `enrole.auditing.pageSize=10`
`enrole.auditing.pageLinkMax`	
	Specifies the number of page links for multi-page result sets on the failed activity. Example (default): `enrole.auditing.pageLinkMax=10`
`enrole.auditing.viewRequests.skipServiceLookup.customProcessTypes`	
	Do not change this property key and value unless you are a qualified administrator. Specifies the custom process type that does not have a service or an account as subject data in the input parameters of its corresponding workflow operation. To use this property, add it to the *$ISIM_HOME*`/data/enroleAuditing.properties` file with a custom process type value. Valid values: A comma-separated custom process type value. Example (default): `enrole.auditing.viewRequests.skipServiceLookup.customProcessTypes=CP`

enRoleAuthentication.properties

The `enRoleAuthentication.properties` file specifies the type of method that is used by the Security Identity Manager Server to authenticate users and identifies the Java object that provides the specified authentication mechanism.

Additionally, the file specifies objects that support IBM Security Access Manager WebSEAL single sign-on and administration of IBM Security Identity Manager to managed remote services.

Authentication properties are specified with a property key and value pair format:

property-key-name=value

The *property-key-name* is an identifier for the authentication mechanism or resource. The *value* is the name of the Java object that provides the authentication service, expressed also as a key and value pair.

`factory=`*value*

The **factory** key name represents a special category for authentication support within the IBM Security Identity Manager software. The *value* is the actual name of the Java object.

For example (entered on one line):

```
enrole.authentication.provider.service=
    factory=com.ibm.enrole.authentication.service.
    ServiceAuthenticationProviderFactory
```

Table 17 defines the properties used to configure IBM Security Identity Manager authentication.

Table 17. enRoleAuthentication.properties properties

Authentication method	
`enrole.authentication.requiredCredentials={simple}`	
	Specifies the required authentication method for users who log in to the IBM Security Identity Manager Server. The valid value for this property is: • **simple** - User name and password. Example (default): `enrole.authentication.requiredCredentials=simple`
Authentication providers (factories)	
`enrole.authentication.provider.simple`	

	Specifies the Java object that handles authentication with user name and password. Custom authentication providers are not supported in the IBM Security Identity Manager Server virtual appliance.
	
	Example (entered on a single line):
	`enrole.authentication.provider.simple=\`
	`factory=com.ibm.itim.authentication.simple.`
	`SimpleAuthenticationProviderFactory`

Authentication service provider	
`enrole.authentication.provider.service`	

	Specifies the Java object that transparently handles IBM Security Identity Manager access to managed remote services and to manage changes in the accounts to these remote services.
	
	These changes include addition, deletion, suspension, restoration, and modification of accounts on the remote service. When you log in to IBM Security Identity Manager, you can change the login and password information for an account on the managed remote service.
	
	The `ServiceAuthenticationProviderFactory` mechanism works with the agent for a given remote service and processes the changed information.
	
	Example (entered on a single line):
	`enrole.authentication.provider.service=\`
	`factory=com.ibm.itim.authentication.service.`
	`ServiceAuthenticationProviderFactory`

WebSEAL single sign-on	

Table 17. enRoleAuthentication.properties properties (continued)

`enrole.authentication.provider.webseal`	

	Specifies the Java object that allows single sign-on in a WebSEAL environment.
	Example (entered on a single line):
	`enrole.authentication.provider.webseal=\`
	`factory=com.ibm.itim.authentication.webseal.WebsealProviderFactory`

`enrole.authentication.idsEqual`	
	Indicates the appropriate algorithm for mapping the IBM Security Access Manager user ID to an IBM Security Identity Manager user ID. An internal identity mapping algorithm is used to ensure the success of the single sign-on operation. Valid values for this property are: • **true** – The Security Access Manager user ID is the same as the IBM Security Identity Manager user ID. • **false** – The Security Access Manager user ID is not the same as the IBM Security Identity Manager user ID. Example: `enrole.authentication.idsEqual=true`

enRoleLogging.properties

The `enRoleLogging.properties` file specifies attributes that govern the operation of the **jlog** logging and tracing API that is bundled with Security Identity Manager.

`jlog` is a logging package for Java. With this package, you can log messages by message type and priority. At run time, you also can control how these messages are formatted and where they are reported.

Table 18 defines the properties that are used to configure IBM Security Identity Manager logging properties and which are available for modification.

Table 18. enRoleLogging.properties properties

General settings	
`logger.refreshInterval`	
	Specifies the refresh interval [in milliseconds] of the logging properties. Example: `logger.refreshInterval=300000`
`logger.msg.com.ibm.itim.security.logChoice`	

Table 18. enRoleLogging.properties properties (continued)

	Specifies the type of authentication attempts to log. Valid values are: • **failure** —- Log authentication failures. • **success** —- Log authentication successes. • **both** —- Log both authentication failures and successes. Example: `logger.msg.com.ibm.itim.security.logChoice=failure`
`logger.msg.com.ibm.itim.security.logging`	
	Specifies whether authentication attempts are logged or not. Valid values are: • **true** —- Log authentication attempts. • **false** —- Do not log authentication attempts. Example: `logger.msg.com.ibm.itim.security.logging=true`
`logger.msg.level`	
	Specifies the logging level for messages. Valid values are: • INFO • WARN • ERROR Example: `logger.msg.level=INFO`
Message logger properties	
`logger.msg.logging`	
	Turns logging on or off for messages. Valid values are: • **true** —- Turns logging on. • **false** —- Turns logging off. Example: `logger.msg.logging=true`
Trace logger properties	
`logger.trace.logging`	

Table 18. enRoleLogging.properties properties (continued)

	Turns trace logging on or off. Valid values are: • **true** —- Turns logging on. • **false** —- Turns logging off. Example: `logger.trace.logging=true`
Applet tracing properties	
`logger.trace.com.ibm.itim.applet.logging`	
	Enables or disables applet trace logging. Example: `logger.trace.com.ibm.itim.applet.logging=true`
Message logging file handler properties	
`handler.file.msg.fileName`	
	Specifies the message log file. Example: `handler.file.msg.fileName=msg.log`
Security logging file handler properties	
`handler.file.security.fileName`	
	Specifies the security log file. Example: `handler.file.security.fileName=access.log`
Trace file handler properties	
`handler.file.trace.fileName`	
	Specifies the trace file name. Example: `handler.file.trace.fileName=trace.log`
FFDC (First-Failure Data Capture) file copy handler properties	
`handler.ffdc.triggerRepeatTime`	
	Specifies the minimum time [in milliseconds] after an initial triggering that the handler responds to subsequent triggering events. Example: `handler.ffdc.triggerRepeatTime=300000`

Table 18. enRoleLogging.properties properties (continued)

`handler.ffdc.fileCopy.triggerFilter`	
	Specifies the filter to control which events trigger an FFDC action. Example: `handler.ffdc.fileCopy.triggerFilter=filter.msgId`
FFDC message id filter properties	
`filter.msgId.msgIds`	
	Specifies the TMS message IDs that trigger the FFDC action. The listed message IDs represent the most severe system errors. Example (on a single line): `filter.msgId.msgIds=CTGIMA401E CTGIMA438W CTGIME013E CTGIME035E CTGIME203E CTGIMF003E CTGIMF011E CTGIMF012E CTGIMF013E CTGIMF014E`
`filter.msgId.msgIdRepeatTime`	
	Specifies the minimum time in milliseconds to wait after a log event is passed with a TMS message ID before it passes another one with the same ID. Example: `filter.msgId.msgIdRepeatTime=300000`
PDXML formatter properties	
`formatter.PDXML.msg.forceAsMessage`	
	Force the message formatter to format all output as message events, regardless of their contents. Example: `formatter.PDXML.msg.forceAsMessage=true`

enrolepolicies.properties

The `enrolepolicies.properties` file provides standard and custom settings that support the functions of the provisioning policy.

Functions supported by this properties file includes:

- Specifying Java classes to process provisioning policy conflicts with join directives
- Specifying default and non-default join directive caching timeouts
- Declaring policy attributes to be ignored during policy compliance validation

A join directive is a set of rules that is used to determine how attributes are handled when a provisioning policy conflicts with another. Join directives use logical constructs to resolve conflicts. Examples include combining all policy attributes (union), with only common attributes (intersection), and resolving conflicts with Boolean `AND` or `OR` logic.

There are 12 types of join directives that you can use. Provisioning policy join directives take effect when more than one provisioning policy is defined for the same user (or group of users) for the same target service, service instance, or service type.

Custom join directives can be defined by writing a custom Java class, adding it to your class path, and then providing the fully qualified Java class name in the policy configuration GUI. If you extend or replace one of the existing join directive classes, you must add the custom property key and value to the `enrolepolicies.properties` file. For example if you developed a new class (`com.abc.TextualEx`) to replace the existing class for textual joins, the registration line is as follows:

```
provisioning.policy.join.Textual= com.abc.TextualEx
```

Table 19 defines the properties used to configure IBM Security Identity Manager policies.

Table 19. enrolepolicies.properties properties

Join directive classes	

`provisioning.policy.join.PrecedenceSequence=com.ibm.itim.policy.join.`
`PrecedenceSequence`
 	
`provisioning.policy.join.Boolean=com.ibm.itim.policy.join.Boolean`
`provisioning.policy.join.Bitwise=com.ibm.itim.policy.join.Bitwise`
`provisioning.policy.join.Numeric=com.ibm.itim.policy.join.Numeric`
`provisioning.policy.join.Textual=com.ibm.itim.policy.join.Textual`
`provisioning.policy.join.Textual.AppendSeparator=<<<>>>`
`provisioning.policy.join.Multivalued=com.ibm.itim.policy.join.Multivalued`	

	Do not modify these property keys and values. Each property key specifies a Java class. It can be used to process the logic of a join directive that is required to resolve a provisioning policy conflict.
Append separator characters	
`provisioning.policy.join.Textual.AppendSeparator`	
	Specifies the character that is used by the textual join directive Java class to separate individual values of a multi-value attribute. Example: `provisioning.policy.join.Textual.AppendSeparator=<<<>>>`
Join directive cache timeouts	
`provisioning.policy.join.defaultCacheTimeout`	
	Specifies the timeout interval [in seconds] between refreshes of the cache that stores default join directive cache values. The default is 86400 seconds, which is 24 hours. Example (default): `provisioning.policy.join.defaultCacheTimeout=86400`
`provisioning.policy.join.overridingCacheTimeout`	

Table 19. enrolepolicies.properties properties (continued)

	Specifies the timeout interval [in seconds] between refreshes of the cache that stores non-default join directive values. The default is 300 seconds, which is 5 minutes. Example: `provisioning.policy.join.overridingCacheTimeout=300`
Account attributes ignored by policy compliance validation	
Excluded generic attributes (default value=1):	

	`nonvalidateable.attribute.eraccountcompliance`
	`nonvalidateable.attribute.eracl`
	`nonvalidateable.attribute.eraccountstatus`
	`nonvalidateable.attribute.erauthorizationowner`
	`nonvalidateable.attribute.erglobalid`
	`nonvalidateable.attribute.erhistoricalpassword`
	`nonvalidateable.attribute.erisdeleted`
	`nonvalidateable.attribute.erlastmodifiedtime`
	`nonvalidateable.attribute.erlogontimes`
	`nonvalidateable.attribute.ernumlogons`
	`nonvalidateable.attribute.erparent`
	`nonvalidateable.attribute.erpassword`
	`nonvalidateable.attribute.erservice`
	`#nonvalidateable.attribute.eruid`
	`nonvalidateable.attribute.objectclass`
	`nonvalidateable.attribute.owner`
	`nonvalidateable.attribute.ercreatedate`
	`nonvalidateable.attribute.erlaststatuschangedate`
	`nonvalidateable.attribute.erpswdlastchanged`
	`nonvalidateable.attribute.erlastaccessdate`
	`nonvalidateable.attribute.ernumlogonattempt`

Excluded Windows Server attributes:	

	`nonvalidateable.attribute.erntpasswordexpired`
	`nonvalidateable.attribute.erntuserbadpwdcount`
	`nonvalidateable.attribute.erntlockedout`

	Declares account attributes that are to be ignored during policy compliance validation. This exclusion list reduces overhead during compliance validation. It also reduces the risk of system failure that can be caused by attributes that cannot logically be resolved during validation.
Partition size	
`policy.partition.size`	

Table 19. enrolepolicies.properties properties (continued)

	To analyze many persons during a policy change event without incurring transaction timeouts, you must break apart or partition the total number of affected persons. It is done, not for starting the concurrent policy analysis, but strictly to avoid waiting in a single database transaction for all persons to be processed. Creating multiple transactions or quickly partitioning the total number of users diminishes the chance of any (smaller) transactions to exceed the transaction timeout value. When a Application server cluster is used with IBM Security Identity Manager, it is helpful to note that partitioning operation itself is not clustered. It is done on the same Application server node which receives the policy change request. Specifies the number of persons or accounts to be evaluated in each thread during high volume policy analysis. High volume policy analysis occurs when a policy change or a service enforcement level change affects a large group of persons or accounts. A larger partition size results in fewer threads. A smaller partition size results in more executed threads in parallel, which requires more memory. Example (default): `policy.partition.size=2500`
`policy.message.size`	
	Specifies the number of persons that are analyzed as part of policy change within a single JMS message. Since Application server polled and reuses threads, the JMS mechanism queues the individual units of analysis work for all assigned Application server threads or message consumers. It is likely that during large policy changes that affect large numbers of people, all JMS consumer threads are busy processing policy analysis and enforcement; the queue for each thread is saturated with more messages to process. Example (default): `policy.message.size=25`
Additional properties	
`policy.analysisservicebatch.size`	
	Specifies the maximum number of services to be analyzed in each policy analysis message. This property is useful during policy/person analysis when a person has many accounts. To prevent system from running into OOM or hung threads this property can be tuned. By default, the property is commented out and an internal hardcoded value of 100 is applied. This default 100 service per batch is found to be optimal for environments that has users who own up to 50 K accounts across multiple platforms. Example (default): `policy.analysisservicebatch.size=100`
`policy.service.selection.maxsearch.size`	

Table 19. enrolepolicies.properties properties (continued)

	This property is used to return the specified number of Persons that are affected by the policy. It also checks whether the policy references any Person of given user class in any one of its memberships. The number is for person search, which is per policy, and thus additive based on the policies involved. It prevents an accidental explosion of the server's JVM with an OOM. By default, the property has an internal hardcoded value of 10000. This property is used while evaluating a collection of service move operations for persons that are affected by adding a host selection policy. It is also used while evaluating a service selection script. Example (default): `policy.service.selection.maxsearch.size=10000`
`policy.cleanup.commitFrequency`	
	Specifies the number of rows that are to be deleted as a batch from database tables while the policy analysis data is cleaned up.. The value of this property if set to `0`, commits database updates only at the end when the entire cleanup activity is completed. If this property value is set to any number greater than 0, the commit is done when the number of uncommitted database updates are equal to this set value. If negative or non-integer value is specified, then default value of 0 is used. The default value of the property is `0` and suggested values are multiples of 1000 (Ex: 25000). Example (default): `policy.cleanup.commitFrequency=0`
During a provisioning policy preview operation, IBM Security Identity Manager evaluates and joins other dependent provisioning policies that are applicable to a user. Performing a lookup for the policy and its dependent data in directory server and parsing it for each user can hamper performance. IBM Security Identity Manager caches the already parsed policies for better performance. Following caches are created: • Policy Cache: Maintains a mapping of provisioning policy DNs and policy objects in cache with policy DN as the key. • RoleDN Cache: Maintains a mapping of Organizational Role DN and a set of provisioning policy DNs in cache with Organizational Role DN as the key. • ServiceDN Cache: Maintains a mapping of Service DN and a set of provisioning policy DNs in cache with Service DN as the key. The greater number of data objects in the cache, the greater is the consumption of memory. The following three properties help to tune the caches by defining the maximum number of policies, service DNs, and role DNs to be cached.	
Provisioning policy cache size	
`policy.policiescache.size`	

Table 19. enrolepolicies.properties properties (continued)

	Specifies the number of provisioning policies to be cached in Policy cache for each provisioning policy preview request. For better performance, the size can be set to number of policies in the Organization. Example (default): `policy.policiescache.size=100`
Organizational Role DN cache size	
`policy.roledncache.size`	
	Specifies the number of role DNs to be cached in RoleDN cache for each provisioning policy preview request. For better performance, the size can be set to number of roles in the Organization. Example (default): `policy.roledncache.size=100`
Service DN cache size	
`policy.servicedncache.size`	
	Specifies the number of Service DNs to be cached in ServiceDN cache for each provisioning policy preview request. For better performance, the size can be set to number of services in the Organization. Example (default): `policy.servicedncache.size=100`

enroleStartup.properties

The `enroleStartup` file is used to specify startup activities in the Application server environment.

Table 20 defines the properties used to configure IBM Security Identity Manager policies.

Table 20. enroleStartup.properties properties

`enrole.startup.names`	
	Lists the background services that are started during IBM Security Identity Manager startup. Do not modify this property.
`enrole.startup.shutdownTrigger.name`	
	The registered class used during shutdown of processes. Do not modify this property.
`enrole.startup.WAS50J2EEShutdownTrigger.attributes`	
	Additional parameters to be passed in to the registered shutdown class. Do not modify this property.

Table 20. enroleStartup.properties properties (continued)

These properties define the background services startup. Do not modify these properties.	
	enrole.startup.Scheduler.attributes enrole.startup.PasswordExpiration.attributes enrole.startup.DataServices.attributes enrole.startup.PostOffice.attributes enrole.startup.RemotePending.attributes enrole.startup.PolicyAnalysis.attributes enrole.startup.ReconcilerCleanup.attributes enrole.startup.PasswordSynchStore.attributes enrole.startup.Monitoring.attributes enrole.startup.WebServices.attributes
enrole.startup.MessageListeners.attributes	
	The JMS queue endpoint listeners can be deactivated during startup for a node in a cluster with disaster recovery configuration. Do not modify this attribute in a single server setup. Deactivating endpoint listeners can cause JMS queue errors if none of the messages is being processed.
enrole.appServer.standby	
	Defines whether the node that is participating in a cluster setup should be a standby node. A standby node does not participate in background shared workload. Available for cluster setup. Do not modify this attribute in a single server setup.
enrole.appServer.standby.inactiveMessageListeners	
	Provides an override to the list of message endpoint listeners to be deactivated in a standby mode. Effective only when enrole.appServer.standby is true.
enrole.appServer.standby.inactiveStartupInitializer	
	Provides an override to the list of background services to be deactivated in a standby mode. Effective only when enrole.appServer.standby is true.

enroleworkflow.properties

The enroleworkflow.properties file specifies the XML file mappings for system-defined workflows.

A workflow is a process that specifies the flow of operations that involve business operations and human interactions. A workflow design defines the manner in which a particular business logic is processed. The XML files specified in the enroleworkflow.properties file implement workflow designs.

The system workflow is identified by a unique type ID and an associated XML file. The XML workflow files are in the following directory:

\data\workflow_systemprocess

Do not remove or modify the default system workflow type IDs and XML file values in the enroleworkflow.properties file.

The updating of the following XML files is not supported.

Table 21 defines the properties used to configure IBM Security Identity Manager workflows.

Table 21. enroleworkflow.properties properties

Policy enforcement workflow
`enrole.workflow.PS=enforcepolicyforservice.xml`
Account fulfillment for noncompliant accounts workflow
`enrole.workflow.EN=fulfillpolicyforaccount.xml`
Service selection management workflow

`enrole.workflow.SA=addserviceselectionpolicy.xml`
`enrole.workflow.SC=changeserviceselectionpolicy.xml`
`enrole.workflow.SD=removeserviceselectionpolicy.xml`

Provisioning policy management workflow

`#Add policy`
`enrole.workflow.PA=addpolicy.xml`
`#Modify policy`
`enrole.workflow.PC=changepolicy.xml`
`#Delete policy`
`enrole.workflow.PD=removepolicy.xml`
`#User BU change`
`enrole.workflow.UO=userbuchange.xml`

Reconciliation workflow
`enrole.workflow.RC=reconciliation.xml` `enrole.workflow.HR=hrfeed.xml`
Dynamic role workflow

`#Add dynamic role`
`enrole.workflow.DA=adddynamicrole.xml`
`#Modify dynamic role`
`enrole.workflow.DC=changedynamicrole.xml`
`#Delete dynamic role`
`enrole.workflow.DD=removedynamicrole.xml`
`#Import Policy Enforcement`
`enrole.workflow.PE=importpolicyenforcement.xml`
`#Process Lifecycle Rule`
`enrole.workflow.LC=lifecyclerule.xml`

helpmappings.properties

The `helpmappings.properties` file allows a customer to replace the installed Security Identity Manager help system with an alternative help system.

The helpmappings.properties file contains the following properties:

Table 22. helpmappings.properties properties

`url.contexthelp`

Table 22. helpmappings.properties properties (continued)

	Specifies an external URL for help. The default is blank, which uses the URL of the IBM Security Identity Manager help system. The URL will also add the resolved locale based on the IBM Security Identity Manager language packs that are installed. For example, http://www.example.com/help/en/ui_login.html Example: `url.contexthelp=www.example.com/help` Clicking on the help icon ('?') in the IBM Security Identity Manager graphical user interface will load the html file from the key mapping (http://www.example.com/help/*customerfilename*.html). For a login page, the value of *customerfilename* might be ui_login.html, and the full address might be http://www.example.com/help/ui_login.html.

reportingLabels.properties

This properties file is like other labels-related properties files such as `labels.properties`, or `customLabels.properties`, and holds labels that are used by Reports.

reporttabledeny.properties

By default, this property holds a list of Security Identity Manager tables that are used by various Security Identity Manager components to store internal or configuration data that is inappropriate for a report.

This file is used by IBM Security Identity Manager Server for Reporting Engine purposes.

The following table defines the properties that determine which information is not exposed in reports.

Table 23. reporttabledeny.properties

`tables`	
	Holds a comma-separated list of all IBM Security Identity Manager database tables that are excluded from report production. If a table is part of this property, the table and its columns are not in the Report Designer; a report cannot be designed on columns of this table. A user who wants to deny a specific database table from being used by the Report Designer can choose to add the table against the `tables` property. Example: `tables=JMSState, JMSStore, entity_column, column_report, report, synchronization_history, synchronization_lock, changelog, resources_synchronizations, NextValue, ListData, AUTH_KEY, ATTR_CHANGE, ACCT_CHANGE, LCR_INPROGRESS_TABLE, WORKFLOW_CALLBACK, POLICY_ANALYSIS, POLICY_ANALYSIS_ERROR, PO_TOPIC_TABLE, PO_NOTIFICATION_TABLE, BULK_DATA_SERVICE, MIGRATION_STATUS, SYNCH_POINT, COMPLIANCE_ALERT, PO_NOTIFICATION_HTMLBODY_TABLE, BULK_DATA_STORE, BULK_DATA_INDEX, MANUAL_SERVICE_RECON_ACCOUNTS, SCRIPT, ACTIVITY_LOCK`
`allowedRestrictedColumns`	

Table 23. reporttabledeny.properties (continued)

	Allows IBM Security Identity Manager administrators to explicitly allow columns of restricted data types, to be used for designing and running custom reports. Such reports however work for IBM Security Identity Manager Administrators only. If a non-administrator attempts to run such reports, the user receives an `AuthorizationException`.
	
	By default, columns of the following restricted data types are not available when you design or run custom reports:
	`BLOB, CLOB, BINARY, VARBINARY, LONGVARBINARY and LONGVARCHAR`
	
	The value of the property is a comma-separated list of `<TABLE_NAME>.<COLUMN_NAME>`. If this property is undefined, then none of the columns of the restricted data type is available for reporting.
	
	Example (on a single line):
	`allowedRestrictedColumns=ACTIVITY.RESULT_DETAIL, PROCESS.RESULT_DETAIL, PROCESSLOG.NEW_DATA`

scriptframework.properties (Suggested)

For *all* new JavaScript extensions, use the `scriptframework.properties` file to configure script extensions and other scripting functions.

JavaScript is used in IBM Security Identity Manager to specify identity policies, provisioning policy parameters, service selection policies, placement rules for identity feeds, and orphan account adoption.

In addition, JavaScript is used in workflows to specify transition conditions, loop conditions, JavaScript activities, activity postscripts, and workflow notification. Various scripting extensions are provided by IBM Security Identity Manager to expose useful data and services to each of these scripts. In addition to these extensions, system administrators can configure IBM Security Identity Manager to load custom JavaScript extensions.

The file `scriptframework.properties` is used to configure all parts of scripting support in IBM Security Identity Manager. It includes which script extensions to use, which script interpreter to use, and other properties that relate to scripting.

The major parts of the `scriptframework.properties` are divided by these host components: `PostOffice`, `ProvisioningPolicy`, `AccountTemplate`, `HostSelection`, `PersonPlacementRules`, `Workflow`, `Reminder`, `IdentityPolicy`, `Notification`, and `OrphanAdoption`.

The most heavily used section of the property file is for configuring which extensions to load for each host component. To have the script framework load an extension, add a key-value line to the `scriptframework.propertiesfile` that is similar to this example:

`ITIM.extension.{`*`Host Component`*`}=`*`com.ibm.itim.class_name`*

where `ITIM.extension.{`*`Host Component`*`}`is the key and *`com.ibm.itim.class_name`* is the value. The value of {*`Host Component`*}can be any of the previously listed components. If you want to load more than a single extension for a host component, you can add a suffix to host component, such as:

`ITIM.extension.{`*`Host Component`*`}.suffix=`*`com.ibm.itim.class_name`*

The only rule is that each key must be unique in the file.

The `scriptframework.properties` file comes pre-configured to load the extensions necessary to use IBM Security Identity Manager with its default scripts. Do not remove any lines in `scriptframework.properties` because removal might cause IBM Security Identity Manager to stop functioning properly.

The next section of the `scriptframework.properties` file configures which script interpreter to use for each host component. IBM Security Identity Manager currently supports two different script interpreters, the IBM JSEngine and the FESI JavaScript Interpreter.

To configure which interpreter to use for each host component, there is a line in the file that looks like:

`ITIM.interpreter.{`*`Host Component`*`}={`*`Engine`*`}`

The value of {*`Host Component`*}can be any of the previously listed components. The value of {*`Engine`*}can be either IBMJS or FESI. The {*`Engine`*}variable is not case-sensitive, so typing `fesi` works as well as typing `FESI`. IBMJS is the default scripting engine, so any value for {*`Engine`*} other than IBMJS or FESI, or no value, uses the IBMJS engine. The FESI engine is deprecated. Use it only if you upgraded from IBM Security Identity Manager Version 4.6 or earlier and have custom FESI extensions.

The next section in the configuration file enables configuring custom JavaScript wrappers. For security reasons, IBM Security Identity Manager does not expose all objects to the scripting environment. Instead, most objects are wrapped in a more restrictive wrapper class that exposes only certain methods. IBM Security Identity Manager has a default wrapper configuration that you can override or extend in this section. This feature is for an advanced user; in most cases do not use it. For more details on how to configure custom wrappers, see the comments in the `scriptframework.properties` file.

In the next section, you can configure direct Java access from scripts run by the IBM JSEngine interpreter. Direct Java access is powerful, but scripts can bypass some of the security built into the script framework. Consider carefully before you do so. See the comments in the `scriptframework.properties` file for more information about how to enable direct Java access.

The final section of the configuration file configures specific properties that might be useful. Each specific property is explained in comments in the `scriptframework.properties` file, including default and allowed values.

Error handling

To enable retry of an activity in the workflow, when the script evaluation fails, due to Directory Server being down, add the following new property:

```
ITIM.script.extension.retryOn=standardException.jndiCommunicationException,
standardException.jndiServiceUnavailable
```

Additionally any custom IBMJS code, used in the workflow, which performs any LDAP operation, must be modified to throw `ScriptException`, with the exception message set to either `standardException.jndiCommunicationException`, or `standardException.jndiServiceUnavailable`".

For example:

```
...
try {
//Custom Script Code performing LDAP operation
} catch(ModelCommunicationException e) {
String msg = "standardException.jndiCommunicationException";
throw new ScriptException(msg, e);
}
...
```

Alternatively, the exception message that is thrown by a custom script extension, can be included as another value for 'ITIM.script.extension.retryOn' property. Multiple values that are specified for this property must be comma separated.

SelfServiceHelp.properties

The SelfServiceHelp.properties file can be used to redirect help to a custom location for customers who want to have their own help content for the self-service user interface.

Table 24 defines the properties used to redirect help to a custom location.

Table 24. SelfServiceHelp properties

IBM Security Identity Manager SelfServiceHelp settings	
helpBaseUrl	
	Specifies the base url to send help requests to. A blank value indicates that help goes to the URL for Self Service application help. Valid values include the URL of the Self Service application help. Example: helpBaseUrl=http://myserver:80
Help Id mappings include: *helpId = relative page URL*	
	The help mappings section maps ids from specific pages to a relative URL sent to the help server. For example: helpBaseUrl=http://myserver:80 locale = en_US loginId/relativeURL = login_help_url=ui/ui_eui_login.html Final URL = http://myserver:80/en_US/ui/ui_eui_login.html Locale is determined by resolving the SelfServiceScreenText.properties resource bundle for the current logged in user and with the associated locale.

SelfServiceHomePage.properties

The SelfServiceHomePage.properties file is used to configure the sections of the initially installed home page for the self-service user interface. You can add or remove tasks, and update icon URLs and labels of the home page from this file.

The file has these types of entries:

- Sections=ActionNeeded, Password, *sectionConfigName* ...

  Defines the section configuration names in the order in which they are displayed.

- Section definition

  Defines the label keys, icons, and other objects for the home page section.
- Task definitions

  Defines the NLS key and link for the URL, the NLS key for the task description, and other attributes that enable displaying the task.

For more information about these properties, see documentation in the properties file.

SelfServiceUI.properties

The `SelfServiceUI.properties` file controls miscellaneous properties of the self-service user interface.

Table 25 defines the properties used to configure the self-service user interface.

Table 25. SelfServiceUI. properties

`enrole.ui.pageSize`	
	Specifies the page size for displaying lists. Example: `enrole.ui.pageSize=10`
`enrole.ui.pageLinkMax`	
	Specifies the number of page links to be shown for multi-page result sets. Example: `enrole.ui.pageLinkMax=100`
`enrole.ui.maxSearchResults`	
	Specifies the maximum number of items returned from a search. The results that are returned can be less than, but not larger than the values specified in `ui.properties`. Example: `enrole.ui.maxSearchResults=1000`
`enrole.ui.maxSearchResults.users`	
	Specifies the maximum displayable search results for the task **Delegate Activities - Search for User**. Example: `enrole.ui.maxSearchResults.users=100`
`enrole.ui.maxNrOfIteration`	
	Specifies the maximum number of wait iterations for `RequestInfo` status. Example: `enrole.ui.maxNrOfIteration=20`
`enrole.ui.waitTime`	
	Specifies the time to wait until the request is asked for next status. The product of (`maxNrOfIteration * waitTime`) is a maximum of 60 seconds. The value is interpreted in milliseconds. Example: `enrole.ui.waitTime=3000`
`enrole.ui.logoffURL`	

Table 25. SelfServiceUI. properties (continued)

	Specifies the URL to forward the browser to when the user logs off. Example: `enrole.ui.logoffURL=myLogoffURL`
`enrole.ui.timeoutURL`	
	Specifies the URL to which forward the browser on timeout. Example: `enrole.ui.timeoutURL=myTimeoutURL`
`ui.layout.showBanner`	
	Specifies a change to the values of `ui.layout` properties to show or hide the banner of the self-service user interface. Example: `ui.layout.showBanner=true`
`ui.layout.showFooter`	
	Specifies a change to the values of `ui.layout` properties to show or hide the footer of the self-service user interface. Example: `ui.layout.showFooter=true`
`ui.layout.showToolbar`	
	Specifies a change to the values of `ui.layout` properties to show or hide the toolbar of the self-service user interface. Example: `ui.layout.showToolbar=true`
`ui.layout.showNav`	
	Specifies a change to the values of `ui.layout` properties to show or hide the page navigation of the self-service user interface. Example: `ui.layout.showNav=false`
`ui.usersearch.attr.cn`	
	Specifies the attribute that is listed in the **searchBy** field for a user search. The attribute is prefixed with `ui.usersearch.attr`. For more information about mapping and syntax, see the documentation in the `SelfServiceUI.properties` file. Example: `ui.usersearch.attr.cn=cn`
`ui.usersearch.attr.sn`	
	Specifies the attribute that is listed in the **searchBy** field for a user search. The attribute is prefixed with `ui.usersearch.attr`. For more information about mapping and syntax, see the documentation in the `SelfServiceUI.properties` file. Example: `ui.usersearch.attr.sn=sn`
`ui.usersearch.attr.telephonenumber`	

Table 25. SelfServiceUI. properties (continued)

	Specifies the attribute that is listed in the **searchBy** field for a user search. The attribute is prefixed with `ui.usersearch.attr`. For more information about mapping and syntax, see the documentation in the `SelfServiceUI.properties` file. Example: `ui.usersearch.attr.telephonenumber=telephonenumber`
`ui.usersearch.attr.mail`	
	Specifies the attribute that is listed in the **searchBy** field for a user search. The attribute is prefixed with `ui.usersearch.attr`. For more information about mapping and syntax, see the documentation in the `SelfServiceUI.properties` file. Example: `ui.usersearch.attr.mail=mail`
`ui.view.accounts.expandedbydefault`	
	Specifies whether the accounts affected twistie state on the change password page are expanded or collapsed (`true|false`) by default. Valid values are: • **true** – Expand the accounts affected twistie state on the change password page by default • **false** – Do not expand the accounts affected twistie state on the change password page by default Example (default): `ui.view.accounts.expandedbydefault=false`
`ui.select.all.accounts`	
	Specifies whether all the accounts under the account twistie are to be selected by default. Valid values are: • **all** – To select all the accounts under the account twistie • **none** – To select none of the accounts under the account twistie • **default** – To retain the default behavior Example (default): `ui.select.all.accounts=default`

ui.properties

The `ui.properties` file specifies attributes that affect the operation and display of the Security Identity Manager graphical user interface.

The following table defines the properties for configuring the IBM Security Identity Manager graphical user interface.

Table 26. `ui.properties` properties

IBM Security Identity Manager GUI configuration settings
`enrole.ui.customerLogo.image`

Table 26. `ui.properties` properties (continued)

	Specifies the file name of the graphic that is displayed on the right side of the IBM Security Identity Manager title banner. The graphic is usually a company logo. For display over the web in a browser, the format of the file must be type that the browser supports. The actual graphics file must be stored in the following location:
	`directories/itim_console.war/html/images/`
	To store the files, use the Custom File Management page from the **Appliance Dashboard** of the IBM Security Identity Manager virtual appliance console. See Managing custom files.
	You can also specify a value, `ibm_banner.gif` to update it.
	Example:
	`enrole.ui.customerLogo.image=ibm_banner.gif`
	Use the Update Property page from the **Appliance Dashboard** of the IBM Security Identity Manager virtual appliance console. See Managing the server properties.

`enrole.ui.customerLogo.url`	
	Specifies the URL link that is activated when you click the custom graphic image (company logo) on the right side of the IBM Security Identity Manager banner. Example: `enrole.ui.customerLogo.url=www.ibm.com`
`enrole.ui.pageSize`	
	Specifies the number of list items that is initially displayed on the screen. If there are more items in the list, links are at the bottom of the list view that activate continuations of the list. For example, Page 2, Page 3, Page 4. Example: `enrole.ui.pageSize=50`
`enrole.ui.maxSearchResults`	
	Specifies the maximum number of items that are returned for a search. This property limits the number of items that are returned when a search is done on the directory server. The evaluation of the ACIs is done later on these returned items. The number of items in the directory server is greater than the value specified for this property. So, the number of items that are displayed on the IBM Security Identity Manager Console might be less than the value specified. The value for this property can control possible system performance degradation when a large return of items is encountered. If you modify the value for this property, you must restart the application server. Example: `enrole.ui.maxSearchResults=1000`
`ui.banner.showForLogin`	
	Specifies whether to show the console banner on the login page, rather than the default login banner. Any customization to the console banner is also on the login page when this property is in effect. **yes** Show the console banner in the login page. **no** Show the default login banner. An empty value assumes no. Example (default): `ui.banner.showForLogin=no`
`ui.footer.URL`	

Table 26. ui.properties properties (continued)

	Specifies the URL for the IBM Security Identity Manager Console. Specify either the full address (`http://yourhost.com/footer.html`) or an address from the IBM Security Identity Manager web server (`/itim/console/custom/footer.html`). A blank value uses the default address of the IBM Security Identity Manager footer. Example: `ui.footer.URL=http://itim99.mylab.raleigh.ibm.com:9080/itim/console/main` Use the Update Property page from the **Appliance Dashboard** of the IBM Security Identity Manager virtual appliance console. See Managing the server properties.
`ui.footer.height`	
	Specifies the height in pixels of the footer on the IBM Security Identity Manager Console. Example (default): `ui.footer.height=50`
`ui.footer.isVisible`	
	Shows or hides the footer of the IBM Security Identity Manager Console. Valid values are as follows: **yes (or blank)** Shows the footer. **no** Hides the footer. Example (default): `ui.footer.isVisible=yes`
`ui.banner.URL`	
	Specifies the URL for the banner on the IBM Security Identity Manager Console. Specify either the full address (`http://yourhost.com/banner.html`) or a path from the IBM Security Identity Manager web server (`/itim/console/custom/banner.html`). A blank value uses the default address of the IBM Security Identity Manager banner. Example: `ui.banner.URL=http://itim99.mylab.raleigh.ibm.com:9080/itim/console/main`
`ui.banner.height`	
	Specifies the height in pixels of the banner on the IBM Security Identity Manager Console. Example (default): `ui.banner.height=48`
`ui.homepage.path`	

Table 26. `ui.properties` properties (continued)

	IBM Security Identity Manager Console home page location. Specify a relative path from the IBM Security Identity Manager Console context root (`/itim/console`). For example, if the full path to the home page was `http://yourhost:9080/itim/console/custom/home.html`, then the following value is `ui.homepage.path=custom/home.html`. The custom home page must be in the IBM Security Identity Manager web application. For example: *path*`/ITIM.ear/itim_console.war/custom/home.html`). A blank value uses the default address of the IBM Security Identity Manager home page. Example: `ui.homepage.path=custom/home.html`
`ui.titlebar.text`	
	Specifies the text in the title bar of the browser for the IBM Security Identity Manager Console. A blank value uses the default name of the IBM Security Identity Manager product. Example: `ui.titlebar.text=Our Home Page`
`ui.userManagement.includeAccounts`	
	Specifies the default behavior for including accounts when you suspend, restore, or delete users. Valid values are as follows: **true** Accounts are included. **false** Accounts are excluded. Example (default): `ui.userManagement.includeAccounts=true`
`ui.userManagement.search.attributes`	
	Adds a search attribute to the default list for the **Manage Users** page in the IBM Security Identity Manager Console. Provide one or more attribute names in the `ui.userManagement.search.attributes` property value that is separated by a comma. Make sure to provide valid and non-repetitive attributes. Do not specify attributes that cannot be searched by using plain text. For example, audio, photo, and other similar items. Example: `ui.userManagement.search.attributes=homepostaladdress,employeenumber` By default, this property value is empty. The property adds user attributes that display in the **Search By** list on the **Manage Users** page for the person search filter.
`ui.challengeResponse.showAnswers`	

Table 26. `ui.properties` properties (continued)

	Specifies whether the answers to challenge response questions is treated as passwords or as clear text in the IBM Security Identity Manager Console of the following pages: • **Forgot Password** page • Challenge response question and answer definition page Valid values are as follows: **true** Answers to challenge response questions is clear text. **false** Answers to challenge response questions is treated as passwords. Example (default): `ui.challengeResponse.showAnswers=true`
`ui.challengeResponse.bypassChallengeResponse`	
	Specifies whether the challenge response questions can be bypassed when the user first logs on to the IBM Security Identity Manager Console, the self service web user interface, or the Identity Service Center. Valid values: **true** When true, the user can cancel and not answer the challenge questions. **false** When false, the user cannot cancel. The user is forced to respond to the challenge questions. Default value: true Example: `ui.challengeResponse.bypassChallengeResponse=true`
`ui.viewAllRequests.loadDefaultQueryResult`	
	Specifies whether the **View All Requests** page loads the default query result. **true** Loads the **View All Requests** page with default query result. **false** Does not load the **View All Requests** page with default query result. Default value: `false` Example: `ui.viewAllRequests.loadDefaultQueryResult=false`
`ui.allowLaunchingNewTaskWithoutWarningForActiveTask`	
	Specifies whether to start selected task or not, if the same task is already active in the IBM Security Identity Manager Console. The examples of the tasks are as follows: `Create Service`, `Change Service`, `Create User`, `Change User`. **true** When you try to start an already active task, the existing task is closed. Starts the new task without displaying any warning message. **false** When you try to start an already active task, a warning message is displayed. Does not start the new task. Default value: `false` Example: `ui.allowLaunchingNewTaskWithoutWarningForActiveTask=false`
`ui.policyManagement.manageProvisioningPolicies.create.defaultMemberType`	

Table 26. `ui.properties` properties (continued)

	Controls default selection of policy membership. This property allows default member type to be selected while you create a provisioning policy. Allowed values are as follows: **users** All users in the organization. **roles** Roles that are specified later. **others** All other users who are not granted to the entitlements that are defined by this provisioning policy by way of other policies. Default value: `users` Example: `ui.policyManagement.manageProvisioningPolicies.create.defaultMemberType= users`
`ui.manageServices.reconcileNow.defaultSelectQuery`	
	Specifies the default reconciliation query option. Allowed values are as follows: **none** None. **use_query** Use query from existing schedule. **define_query** Define query. Default value: `none` Example: `ui.manageServices.reconcileNow.defaultSelectQuery=none`
`ui.passwordManagement.defaultSelection.typePassword`	
	Specifies **Allow me to type a password** as default over the current **Generate a password for me** option. Allowed values are as follows: **true** Selects the **Allow me to type a password** option and additionally none of the accounts get selected by default. **false** Selects the **Generate a password for me** option if this property is set to false or not present. Default value: `false` Example: `ui.passwordManagement.defaultSelection.typePassword=false`
`ui.advancedUserSearch.AllTypes.defaultSearchAttribute.names` `ui.advancedUserSearch.AllTypes.defaultSearchAttribute.labels`	

Table 26. ui.properties properties (continued)

	When you select **User type** as **All types** in the **Select User Type** page, the properties add the default search attributes and its labels on the **Advanced Search** page for users in the IBM Security Identity Manager Console. If the `ui.advancedUserSearch.AllTypes.defaultSearchAttribute.names` property is removed or if no value is specified, then IBM Security Identity Manager does not display any default search attribute field.
	
	Provide one or more attribute names in the `ui.advancedUserSearch.AllTypes.defaultSearchAttribute.names` property value, and corresponding attribute labels in the `ui.advancedUserSearch.AllTypes.defaultSearchAttribute.labels` property value.
	
	Make sure to provide valid, non-repetitive, and comma-separated values. Do not specify attributes that cannot be searched by using plain text. For example, audio, photo, and other similar items.
	
	Example (default):
	`ui.advancedUserSearch.AllTypes.defaultSearchAttribute.names=cn`
	`ui.advancedUserSearch.AllTypes.defaultSearchAttribute.labels=$cn`
	
	The property adds the default search attributes and its labels on the **Advanced Search** page for users when you select **User type** as **All types** in the **Select User Type** page.

WfDesigner and FormDesigner applet properties	

`enrole.build.version`
`enrole.java.plugin`
`enrole.java.plugin.classid`
`enrole.java.pluginspage`
`enrole.java.plugin.jpi-version`
`enrole.java.plugin.version`
`enrole.java.entWflowHeightIE`
`enrole.java.entWflowWidthIE`
`enrole.java.entWflowHeightMZ`
`enrole.java.entWflowWidthMZ`
`enrole.java.opWflowHeightIE`
`enrole.java.opWflowWidthIE`
`enrole.java.opWflowHeightMZ`
`enrole.java.opWflowWidthMZ`
`enrole.java.joinDirHeightIE`
`enrole.java.joinDirWidthIE`
`enrole.java.joinDirHeightMZ`
`enrole.java.joinDirWidthMZ`
`enrole.java.formDesignHeightIE`
`enrole.java.formDesignWidthIE`
`enrole.java.formDesignHeightMZ`
`enrole.java.formDesignWidthMZ`
`express.java.formDesignHeightIE`
`express.java.formDesignWidthIE`
`express.java.formDesignHeightMZ`
`express.java.formDesignWidthMZ`
`#enrole.ui.logoffURL (default is commented out)`
`#enrole.ui.timeoutURL (default is commented out)`	

	You must not modify or remove any information for these properties in the property file. These property key and value pairs provide the necessary Java applet support required by the Java Web Start that runs the IBM Security Identity Manager Console.

Table 26. `ui.properties` properties (continued)

Report menu properties	
`enrole.ui.report.maxRecordsInReport`	
	Displays the number of records that can be displayed in a PDF report without encountering an "Out of Memory" error. The number does not ensure that PDF report generation is successful. If the report contains more records than specified by this property, PDF report generation is not attempted. Example: `enrole.ui.report.maxRecordsInReport=5000`
Enable or disable WebSEAL single sign-on (SSO)	
`enrole.ui.ssoEnabled`	
	The property key and value pairs do not pertain to the IBM Security Identity Manager Console. Enable or disables WebSEAL single sign-on. More configuration is required for WebSEAL single sign-on. Valid values are as follows: **true** WebSEAL single sign-on is enabled. **false** WebSEAL single sign-on is disabled. Example (default): `enrole.ui.ssoEnabled=false`
`enrole.ui.ssoEncoding`	
	Specifies the encoding that is used to decode user credentials with WebSEAL single sign-on. Example (default): `enrole.ui.ssoEncoding=UTF-8`
Refresh properties	
`enrole.ui.httpRefreshSecs`	
	Defines, in seconds, the refresh rate for pages within the IBM Security Identity Manager Console. This property is used during policy previews. Example (default): `enrole.ui.httpRefreshSecs=10`
Search class mapping for ObjectProfileCategory	
	The property key and value pairs do not pertain to the IBM Security Identity Manager Console and must not be modified or removed.
Justification field configuration properties	
`ui.displayJustification`	

Table 26. ui.properties properties (continued)

	Specifies whether the **Justification** field is displayed in the user interface. By default, the **Justification** field is not displayed. Use in conjunction with the `enrole.justificationRequired` property in the `enRole.properties` file. Example (default): `ui.displayJustification=false`
`Identity Service Center as the default user interface configuration property`	
`ui.defaultui.redirectSelfToISC`	
	Specifies whether the Identity Service Center user interface is set as the default user interface. If a user is already authenticated to the IBM Security Identity Manager, and starts the self-service user interface, no redirection happens. **true** If the Identity Service Center is deployed and if a user starts the self-service user interface, then the self-service user interface redirects the user to the Identity Service Center. **false** When a user starts the self-service user interface, it does not redirect a user to theIdentity Service Center. The self-service user interface starts. Example (default): `ui.defaultui.redirectSelfToISC=false`
`Generate password configuration property`	
`ui.passwordManagement.generatePassword`	
	Specifies which change password options to enable on the Identity Service Center user interface. This property is applicable only when the **Enable password editing** is selected in the administrative console. The valid values are: **true** Enables both the **Generate a password for me** and **Allow me to type a password** options. The `ui.passwordManagement.defaultSelection.typePassword` property is applicable only if the property `ui.passwordManagement.generatePassword` is set to `true`. **false** Enables the **Generate a password for me** option and disables the **Allow me to type a password** option. Example (default): `ui.passwordManagement.generatePassword=true`
`Challenge response answers display configuration property`	
`ui.challengeResponse.showAnswers`	
	Shows or hides the challenge response answers that a user types in the text box. The valid values are: **true** Shows what a user types. **false** Hides what a user types. Example (default): `ui.challengeResponse.showAnswers=true`

UIConfig.properties

The config/UIconfig.properties file contains the several properties that affect the Identity Service Center interface.

Table 27. UIConfig.properties

password.change.pollingTime	
	Specifies in milliseconds the time to wait before checking whether the expired password change request is processed. A value that is less than 0 is invalid. Example (default): password.change.pollingTime=1000
password.change.pollingIterations	
	Specifies the maximum number of times that the server checks whether the password change is processed. A value that is less than 1 is invalid. Example (default): password.change.pollingIterations=5
isim.ui.rtlLocales	
	A comma-separated list of right-to-left locales. The default values are ARABIC(ar) and HEBREW(iw). Example (default): isim.ui.rtlLocales=ar,iw
property.refresh.interval.seconds	
	Defines how frequently the Identity Service Center server refreshes the value of properties by reading the UIConfig.properties file to pick up new values for the changed properties. A user can change this property even while the Identity Service Center server is running. A user does not need to restart the server to pick up the changes. Example (default): property.refresh.interval.seconds=300
LOGO_IMAGE	
	Specifies the file name in custom/ui/images directory that displays the company logo image. Example: LOGO_IMAGE=companyLogo.png
HEADER_LOGO_IMAGE	
	Specifies the file name in custom/ui/images directory that displays the page header logo image. Example: HEADER_LOGO_IMAGE=headerLogo.png
access.selection.maximum.number	

Table 27. UIConfig.properties (continued)

	Specifies the maximum number of accesses that can be selected in the manage access flow. For example, in the Request Access wizard, and Edit and Delete Access wizard. Example (default): `access.selection.maximum.number=25`
`timeout.notify`	
	Specifies the seconds left before the session end that the expiration notification message is sent. Example (default): `timeout.notify=20`
`timeouturl`	
	Specifies the URL to which IBM Security Identity Manager redirects on session timeout Example: `timeouturl=myTimeoutURL`
`ui.activities.approve.justificationRequired`	
	Specifies whether the justification field is mandatory while approving the activity. The valid values are: **true** The justification field is mandatory while approving the activity. **false** The justification field is optional while approving the activity. By default, this property is set to true.
`ui.activities.reject.justificationRequired`	
	Specifies whether the justification field is mandatory while rejecting the activity. The valid values are: **true** The justification field is mandatory while rejecting the activity. **false** The justification field is optional while rejecting the activity. By default, this property is set to true.
`ui.activities.displayJustification`	
	Specifies whether the justification field is displayed in the Manage Activities and Decisions page. **true** The justification field is displayed in the Manage Activities and Decisions page. **false** The justification field is not displayed in the Manage Activities and Decisions page. By default, this property is set to true.
`ui.ssoEnabled`	

Table 27. UIConfig.properties (continued)

	The property key and value pair pertain to the Identity Service Center. The property indicates whether WebSEAL single sign-on is enabled or disabled. To complete the configuration for WebSEAL single sign-on for Identity Service Center, set this property value to `true`. The valid values are: **true** WebSEAL single sign-on is enabled. **false** WebSEAL single sign-on is disabled. By default, this property is set to `false`.
`ui.userPicker.defaultSelfSelect`	
	This property used to specify whether the current logged in user is selected automatically for the Request Access flow and the Edit and Delete Access flow. The valid values are: **true** The current logged in user is selected automatically. **false** No user is selected. By default, this property is set to `false`.

Chapter 16. Select accounts to exclude from reconciliations

Learn how to excluded selected accounts from reconciliations.

To select accounts to exclude from reconciliations, do the following steps:

1. Create an LDIF file and specify the accounts to exclude from reconciliation and the services on which these accounts exist.
2. Import the LDIF file to the LDAP Directory Server.

Exclude accounts from reconciliation

During reconciliations, all accounts are returned from the managed resource, unless otherwise specified by a query. Accounts are automatically adopted if the account is owned by a recognized user in the system or if an alias exists for any account.

However, the IBM Security Identity Manager Server can be configured to prevent automatic adoption of specified accounts. This feature can be used to prevent system accounts, such as `root`, `lp`, `sys`, and `etc` in UNIX resources, from automatically being adopted. This precaution prevents users from accidentally or maliciously adopting and modifying sensitive accounts.

Although these accounts are not automatically adopted, these accounts can still be manually adopted by an administrative user.

The accounts to exclude from reconciliations are specified in an LDIF file. The following excerpt is an example of entries in an LDIF file:

```
dn: ou=excludeAccounts, ou=itim, <TENANT_DN>
ou: excludeAccounts
objectClass: top
objectClass: organizationalunit

dn: cn=SolarisProfile, ou=excludeAccounts, ou=itim, <TENANT_DN>
erObjectProfileName: SolarisProfile
objectClass: top
objectClass: eridentityexclusion
cn: SolarisProfile
erAccountID: root
erAccountID: admin
```

The `cn` and `erObjectProfileName` is the name of the service profile. Excluded accounts are defined by the `erAccountID` attribute. The example excludes the root and admin accounts from automatically being adopted when a reconciliation is run on a Solaris service.

Chapter 17. System property configuration in enRole.properties

Detailed information about the property keys and values that are contained in the `enRole.properties` system configuration file are described here.

The `enRole.properties` system configuration file contains many of the properties used to configure IBM Security Identity Manager. The file properties control the program functions and enable user customization of special features.

From the **Appliance Dashboard** of the IBM Security Identity Manager virtual appliance console, use the Update Property page to work with the `enRole.properties`. See Managing the server properties.

Properties files

Java properties files define attributes that allow customizing and control of the Java software. Standard system properties files and custom properties files are used to configure user preferences and user customization.

A Java properties file defines the values of named resources. It can specify program options such as database access information, environment settings, and special features and functions.

A properties file defines named resources with a property key and value pair format:

property-key-name=value

The *property-key-name* is an identifier for the resource. The *value* is typically the name of the actual Java object. It provides the resource or a String representing the value of the property key, such as `database.name=itimdb`. The statement syntax allows spaces before and after the equal (=) sign. It can span multiple lines if you place a line continuation character \ (a backslash) at the end of the line. For more information about statement syntax, see the Java language references.

Application server properties

Application server properties define values that are specific to integrating IBM Security Identity Manager with the Application server.

Table 28 lists these Application server properties.

Table 28. Application server properties

Platform Context Factory Name
`enrole.platform.contextFactory`

Table 28. Application server properties (continued)

	Do not modify this property key and value.
	
	Specifies the Java class for the platform context factory that defines the integration point for IBM Security Identity Manager with the Application server.
	
	Example (default, entered as a single line):
	`enrole.platform.contextFactory=com.ibm.itim.apps.impl.websphere.`
	`WebSpherePlatformContextFactory`

Application server	
`enrole.appServer.contextFactory`	

	Do not modify this property key and value.
	
	Specifies the Java class that determines which JNDI factory to use with the Application server.
	
	Example (default):
	`enrole.appServer.contextFactory=com.ibm.websphere.naming.`
	`WsnInitialContextFactory`

`enrole.appServer.url`	
	This property key and value can be changed only by a qualified administrator. Specifies the location of the application server naming service. This value is obtained during IBM Security Identity Manager installation. Example: `enrole.appServer.url=iiop://localhost:2809`
`enrole.appServer.usertransaction.jndiname`	
	Do not modify this property key and value. Specifies the JNDI name of the JTA (Java Transaction API) User Transaction object. Example (default): `enrole.appServer.usertransaction.jndiname=jta/usertransaction`
`enrole.appServer.realm`	
	This property key and value can be changed only by a qualified administrator. Specifies the target server security realm name if IBM Security Identity Manager is running on a different Application server instance that is configured to run with different security realm. Example (on a single line): `enrole.appServer.realm=itimCustomRealm` The default value is itimCustomRealm; it can be updated during the installation of IBM Security Identity Manager.
`enrole.appServer.registry`	

Table 28. Application server properties (continued)

	Do not modify this property key and value. Describes the registry to which IBM Security Identity Manager is configured. Example (default): `enrole.appServer.registry=ITIM_Custom_registry`
`enrole.appServer.security.domain`	
	Do not modify this property key and value. Specifies the name of the Security domain that is created for IBM Security Identity Manager. Example (default): `enrole.appServer.security.domain=ISIMSecurityDomain`
`enrole.appServer.alwayssetisolevelrc`	
	Do not modify this property key and value. This property specifies that IBM Security Identity Manager must always set the transaction isolation level to Read-Committed when it acquires database connections. Because the Application server has internal support for setting the isolation level, this property must be set to false. Example (default): `enrole.appServer.alwayssetisolevelrc=false`
Login helper	
`enrole.appServer.loginHelper.class`	
	Do not modify this property key and value. Specifies the Java class that is used to log each thread in to J2EE Security. Example (default): `enrole.appServer.loginHelper.class=com.ibm.itim.util.was.WAS40LoginHelper`
Application server servlet path separator	
`enrole.servlet.path.separator`	
	Do not modify this property key and value. Specifies the separator character that is used to specify path names to required resources. Example (default): `enrole.servlet.path.separator=.`
Event notification system login	
`SystemLoginContextFactory`	

Table 28. Application server properties (continued)

	Do not modify this property key and value. Specifies the Java factory class for event notification system login appropriate for Application server. Example (default, entered as a single line): `SystemLoginContextFactory=com.ibm.itim.remoteservices.provider.itim.websphere.WSSystemLogonContextFactory`
User-selected locale	
`locale`	
	Specifies the locale setting for the IBM Security Identity Manager environment. Example (default): `locale=en`
Context factory name	
`enrole.appServer.name`	
	Specifies the unique name of the application server. In a cluster environment, it is important that this name is unique for each member within a node in the cluster. Cluster members on different nodes can have same names. Example (default): `enrole.appServer.name=myserver`

Remote services properties

The `enrole.remoteservices.assemblyline.encodeusingUTF8` property is referred whenever IBM Security Identity Manager sends the assembly line to IBM Security Directory Integrator dispatcher before running any operation. Use the `UTF-8` encoding when the assembly line contains special characters such as German umlaut characters.

The value of the `enrole.remoteservices.assemblyline.encodeusingUTF8` property determines whether the assembly line sent to IBM Security Directory Integrator is encoded with the `UTF-8` format or not.

Table 29. Remote services properties

`enrole.remoteservices.assemblyline.encodeusingUTF8`

Table 29. Remote services properties (continued)

	Do not change this property key and value unless you are a qualified administrator. Specifies whether the `UTF-8` encoding is used or not. Values include: • **`true`** – Only the `UTF-8` encoding is used. • **`false`** – The platform default encoding is used. Example (default): `enrole.remoteservices.assemblyline.encodeusingUTF8=false`

Web services properties

The web services properties define the properties that are used by IBM Security Identity Manager to manage the web services API.

Table 30 determines the web services properties.

Table 30. Web services properties

`enrole.webServices.version`	
	Do not change this property key. Specifies the web services version. The value is returned by the `WSUnAuthService.getWebServicesVersion` web services API. Values include the version of the web services. Example (default): `enrole.webServices.version=1.0`
`enrole.webseal.ltpa.cookie.name`	
	Do not change this property key and value unless you are a qualified administrator. Specifies the property to identify the name of the HTTP header, which carries the LTPA token. Use this property in SSO mode only. The default value is `LtpaToken2`. Do not change this property unless the HTTP header name that carries the LTPA token is other than the default specified. Example (default): `enrole.webseal.ltpa.cookie.name=LtpaToken2`
`enrole.webServices.session.cache.maxRetry`	
	Do not change this property key and value unless you are a qualified administrator. Use this property key in cluster environment, and when the `enrole.webServices.session.mgmt.clientSide` property is set to false. Values must be a valid integer. Example (default): `enrole.webServices.session.cache.maxRetry=5`
`enrole.webServices.session.mgmt.clientSide`	

Table 30. Web services properties (continued)

	Do not change this property key and value unless you are a qualified administrator. Specifies whether the session management is client side or server side. Values include: • **true** – indicates that client side management is enabled. • **false** – indicates that a server-side management is expected. Example (default): `enrole.webServices.session.mgmt.clientSide=true`
`authTokenTimeout`	
	Do not change this property key and value unless you are a qualified administrator. Specifies the time in hours for how long a session can be valid. For example, even if you keep a session active by continuously using it, the session expires every two days, and you must log in again. Use this property key when the `enrole.webServices.session.mgmt.clientSide` property is set to false. Values include: Example (default): `authTokenTimeout=48`
`sessionInactivityTime`	
	Do not change this property key and value unless you are a qualified administrator. Specifies the time in minutes for how long an unused session is active. Use this property key when the `enrole.webServices.session.mgmt.clientSide` property is set to false. Values include: Example (default): `sessionInactivityTime=15`

Organization properties

Organization properties define the organization name that is used by the directory server.

Table 31 defines the properties for the organization name that is used by the directory server.

Table 31. Organization properties

`enrole.defaulttenant.id`

Table 31. Organization properties (continued)

	Use the **ldapConfig** utility to modify this property.
	Specifies the short format of the organization name that is used by the directory server.
	This value is specified during installation of IBM Security Identity Manager or by running the **ldapConfig** utility.
	Example (default):
	`enrole.defaulttenant.id=org`
	In LDAP, this value is expressed as:
	`ou=org`

`enrole.organization.name`	
	Use the **ldapConfig** utility to modify this property. Specifies the long format of the organization name that is used by the directory server. This value is specified during installation of IBM Security Identity Manager or by running the **ldapConfig** utility. Example (default): `enrole.organization.name=Organization`

LDAP server properties

LDAP server properties define the properties that are used by the directory server in which IBM Security Identity Manager stores data.

Table 32 defines the properties that are used the directory server.

Table 32. LDAP server properties

`enrole.ldapserver.root`	
	Specifies the top-level entry node of the directory server data structure (`dc=domain control`). Use the **ldapConfig** utility to modify this value. This value is specified during installation of IBM Security Identity Manager. Example (default): `enrole.ldapserver.root=dc=com`
`enrole.ldapserver.home`	
	Do not modify this property key and value. Specifies the location of the system configuration information in the directory server. Example (default): `enrole.ldapserver.home=ou=itim`
`enrole.ldapserver.agelimit`	

Table 32. LDAP server properties (continued)

	Do not change this property key and value unless you are a qualified administrator. Use the**runConfig** utility to modify this value. Specifies the number of days that an object remains in the recycle bin before it can be deleted when the cleanup script is started. The recycle bin age limit protects objects in the recycle bin from cleanup scripts for the specified length of time. Cleanup scripts can remove only those objects that are older than the age limit setting. If the age limit setting is 62 days (default), only objects in the recycle bin for more than 62 days can be deleted by starting the cleanup script. Example (default): `enrole.ldapserver.agelimit=62`
`enrole.ldapserver.ditlayout`	

	Do not modify this property key and value.
	
	Specifies the Java class that defines the structure of the data that is stored in the directory server.
	
	Example (default, flat structure):
	`enrole.ldapserver.ditlayout=com.ibm.itim.dataservices.dit.itim.`
	`FlatHashedLayout`

`enrole.ldap.provider`	
	Example (default): `enrole.ldap.provider=IBM`

Search and LDAP control properties

Search and LDAP control properties are used to configure search strategy and LDAP control.

For more information about setting these parameters for your environment, see the tuning guide that is provided for IBM Security Identity Manager.

Table 33 defines the properties used to configure search strategy and LDAP control.

Table 33. Search and LDAP control properties

`enrole.search.sss.enable`	
	Do not modify this property key and value. Specifies whether Server Side Sorting is used for searches of the directory server. Enabling server-side sorting with this property can have a large negative impact when you view large organizational units. It is suggested that you disable this option in most environments. Example (default): `enrole.search.sss.enable=false`
`enrole.search.vlv.enable`	

Table 33. Search and LDAP control properties (continued)

	Do not modify this property key and value. Specifies whether Virtual List View (VLV) is used for all return data from the directory server. This property can be enabled only when supported by the directory server. This option reduces the memory load on the application server but places a significant load on the LDAP server. Example (default): `enrole.search.vlv.enable=false`
`enrole.search.paging.enable`	
	Do not modify this property key and value. Specifies whether Paged Sorting is used for searches of the directory server. This option reduces the memory load on the application server. Enabling it is not suggested because the directory server might place a limit on the number of outstanding paged searches. Example (default): `enrole.search.paging.enable=false`
`enrole.search.paging.pagesize`	
	Do not modify this property key and value. Specifies the page size used for paged LDAP searches when `enrole.search.paging.enable=true`. Example (default): `enrole.search.paging.pagesize=128`
`enrole.search.cache.enable`	
	Do not modify this property key and value. Specifies the use of cached searching to speed up LDAP searches. Example (default): `enrole.search.cache.enable=true`
`enrole.search.cache.secondary.enable`	
	Do not modify this property key and value. Specifies the use of secondary cached searching to speed up LDAP searches. Example (default): `enrole.search.cache.secondary.enable=true`
`enrole.search.cache.secondary.filter.1`	
	Do not modify this property key and value. Use a filter fragment for people to prevent LDAP search filters from getting cached. Filtered out LDAP search filters are cached in the secondary cache, if enabled. Example (default): `enrole.search.cache.secondary.filter.1=ou=people`

Table 33. Search and LDAP control properties (continued)

`enrole.search.cache.secondary.filter.2`	
	Do not modify this property key and value. Use a filter fragment for accounts to prevent LDAP search filters from getting cached. Filtered out LDAP search filters are cached in the secondary cache, if enabled. Example (default): `enrole.search.cache.secondary.filter.2=ou=accounts`
`enrole.search.cache.secondary.filter.3`	
	Do not modify this property key and value. Use a filter fragment for the `systemuser` to prevent LDAP search filters from getting cached. Filtered out LDAP search filters are cached in the secondary cache, if enabled. Example (default): `enrole.search.cache.secondary.filter.3=ou=systemuser`
`enrole.search.cache.secondary.filter.4`	
	Do not modify this property key and value. Use a filter fragment for orphan accounts to prevent LDAP search filters from getting cached. Filtered out LDAP search filters are cached in the secondary cache, if enabled. Example (default): `enrole.search.cache.secondary.filter.4=ou=orphans`
`enrole.search.clientside.filtering.enable`	
	Do not modify this property key and value. Specifies the use of client-side filtering as a performance alternative on complex LDAP searches. Example (default): `enrole.search.clientside.filtering.enable=true`
`enrole.search.strategy`	

Table 33. Search and LDAP control properties (continued)

	Do not modify this property key and value. Specifies the Java class that defines the search strategy to process the return data from the directory server. Strategy values include: • `com.ibm.itim.apps.ejb.search.EnumeratedSearch` (process data on demand) Avoids the use of collections, if possible. Maintains a cache of the number of search links multiplied by the page size. The underlying connection is closed when the page cache is filled. Access control items are applied as results are retrieved. • `com.ibm.itim.apps.ejb.search.CollectedSearch` (process all data) This is the previous search mechanism, which converts the search results into a collection and sort it. Applying access control items on the collection as pages are retrieved. The underlying LDAP connection is freed as soon as the results are transformed into a collection. Example (default): `enrole.search.strategy=com.ibm.itim.apps.ejb.search.EnumeratedSearch`
`enrole.recyclebin.enable`	
	Disable use of the recycle bin for a majority of objects to improve search times. Example (default for new installations): `enrole.recyclebin.enable=false`

Person profile properties

Person profile properties identify a person profile.

Table 34 defines the property used to identify a person profile. This property selects the profile by default when you create people or do advanced person searches in the administrative console.

Table 34. Person profile property

`enrole.personProfile`	
	Searches in IBM Security Identity Manager use the default person profile *Person*. If you want to use custom person schemas, set this property to your profile. Example (default): `enrole.personProfile=Person` Example: `enrole.personProfile=`*`your_profile`*

Profile and schema cache properties

Profile and schema cache properties define system cache performance.

Table 35 defines the properties used to configure system cache performance.

Table 35. Profile and schema cache properties

`enrole.profile.timeout`	
	This property key and value affects performance tuning for IBM Security Identity Manager. Do not change it unless you are a qualified administrator. Specifies the timeout value in minutes for information in the profile section of the cache. Information exceeding this timeout value is removed from the cache. Example (default): `enrole.profile.timeout=10`
`enrole.schema.timeout`	
	This property key and value affects performance tuning for IBM Security Identity Manager. Do not change it unless you are a qualified administrator. Specifies the timeout value in minutes for information in the schema section of the cache. Information exceeding this timeout value is removed from the cache. Example (default): `enrole.schema.timeout=10`
`password.attributes`	
	Specifies which attribute is encrypted by the `dataservices` component. Example (default, on a single line): `password.attributes=ersynchpassword erServicePassword erServicePwd1 erServicePwd2 erServicePwd3 erServicePwd4 erADDomainPassword erPersonPassword erNotesPasswdAddCert eritamcred erep6umds`
`enrole.reminder.timeout`	
	Do not change this property key and value unless you are a qualified administrator. Specifies the cache interval (in minutes) for a workflow reminder. Example: `enrole.reminder.timeout=10`
`signedObjectsCacheTimeout`	
	Do not change this property key and value unless you are a qualified administrator. Specifies the cache interval (in hours) for a signed objects. Example: `signedObjectsCacheTimeout=8`

Messaging properties

Messaging properties configure the internal communication between components of the Java Message Service (JMS) used by IBM Security Identity Manager.

Table 36 defines the properties used to configure the internal communication between components of the Java Message Service (JMS) used by IBM Security Identity Manager.

The adjustment of these property values is important to accurate performance tuning and scalability of the IBM Security Identity Manager product. Do not change property values in this section unless you are a qualified administrator.

Table 36. Messaging properties

Message timeout configuration	
`enrole.messaging.ttl`	
	This property key and value affects performance tuning for JMS. Do not change the value unless you are a qualified administrator. Specifies the lifetime in minutes of a message in the queue. A value of zero specifies an unlimited lifetime. Example (default): `enrole.messaging.ttl=0`
Messaging queue configuration	

`enrole.messaging.managers= \`
`enrole.messaging.adhocSyncQueue \`
`enrole.messaging.workflowQueue \`
`enrole.messaging.sharedWorkflowQueue \`
`enrole.messaging.partitioningServiceQueue \`
`enrole.messaging.remoteServicesQueue \`
`enrole.messaging.remotePendingQueue \`
`enrole.messaging.mailServicesQueue \`
`enrole.messaging.policyAnalysisQueue \`
`enrole.messaging.policySimulationQueue \`
`enrole.messaging.importExportQueue`	

	Do not modify these property keys and values. Specifies the key names of supported IBM Security Identity Manager queues.

`enrole.messaging.adhocSyncQueue=adhocSyncQueue`
`enrole.messaging.workflowQueue=workflowQueue`
`enrole.messaging.sharedWorkflowQueue=sharedWorkflowQueue`
`enrole.messaging.partioningServiceQueue=partitioningServiceQueue`
`enrole.messaging.remoteServicesQueue=remoteServicesQueue`
`enrole.messaging.remotePendingQueue=remotePendingQueue`
`enrole.messaging.mailServicesQueue=mailServicesQueue`
`enrole.messaging.policyAnalysisQueue=policyAnalysisQueue`
`enrole.messaging.policySimulationQueue=policySimulationQueue`
`enrole.messaging.importExportQueue=importExportQueue`	

	Do not modify these property keys and values. Specifies the actual queue name as referenced by the application server.
Queue attribute configuration	

Table 36. Messaging properties (continued)

	• SHARED
	A Boolean value that indicates whether the queue is shared across a clustered deployment. In a cluster, a shared queue can be read and written to by all cluster members.
	Do not modify this property.
	Example (on a single line):
	`enrole.messaging.sharedWorkflowQueue.attributes=SHARED=true`
	`enrole.messaging.policyAnalysisQueue.attributes=SHARED=true`
	`enrole.messaging.policySimulationQueue.attributes=SHARED=true`
	
	Message processing errors detected by the messaging system cause individual messages to be redelivered and additional attempts to handle the message. Following the first indication of process failure, a retry is scheduled immediately. If the first attempt fails, another is scheduled with a delay that matches the value of the FIRST_RETRY_DELAY property. If the second attempt fails, another is scheduled with a delay that matches the value of the RETRY_DELAY property. Subsequent retries are attempted with the value of the RETRY_DELAY property until the MAX_RETRY_TIME threshold is reached.
	
	Set the following properties to manage how the system handles the retry attempts.
	• FIRST_RETRY_DELAY
	The amount of time in milliseconds to delay before retrying after the initial immediate retry. Default value is 900000 (15 minutes).
	• RETRY_DELAY
	The amount of time [in milliseconds] to delay before retrying after the immediate and first attempts fail. Default value is 3600000 (60 minutes).
	• MAX_RETRY_TIME
	The maximum amount of time allowed for attempts, beginning with the first failure. Default value is 86400000 (24 hours)
	
	Example (on a single line):
	`enrole.messaging.workflowQueue.attributes=SHARED=false FIRST_RETRY_DELAY=300000 RETRY_DELAY=900000 MAX_RETRY_TIME=3600000`

Scheduling properties

The scheduling properties are used to configure the internal scheduler that runs calendar-based and scheduled events.

Table 37 defines the properties used to configure the internal scheduler responsible for running calendar-based scheduled events. Events and their schedules are stored in a database table.

Table 37. Scheduling properties

enrole.scheduling.heartbeat	
	This property key and value affects performance tuning for IBM Security Identity Manager. Do not change it unless you are a qualified administrator. Specifies the interval [in seconds] that the event monitor checks the database table for scheduled events. Example (default): `enrole.scheduling.heartbeat=30`

Table 37. Scheduling properties (continued)

enrole.scheduling.timeout	
	This property key and value affects performance tuning for IBM Security Identity Manager. Do not change it unless you are a qualified administrator. Specifies the timeout value [in minutes] for the event processor. Example (default): `enrole.scheduling.timeout=10`
enrole.scheduling.fetchsize	
	This property key and value affects performance tuning for IBM Security Identity Manager. Do not change it unless you are a qualified administrator. Specifies the number of messages to retrieve at a time when in batch mode. Example (default): `enrole.scheduling.fetchsize=50`

Password transaction monitor properties

Password transaction monitor properties checks responses to password transactions. It expires those transactions when the user fails to respond in the specified interval.

When a password for a user is changed or automatically generated, an email notification is sent to a user. The email contains either the actual password or a link that the user can follow to obtain the new password. This activity is called a password transaction. The user must respond to the email and incorporate the new password within a specified amount of time. If the user fails to respond within the allowed time period, the password transaction expires.

The password transaction monitor is responsible for checking responses to password transactions. It expires those transactions when the user fails to respond to the email.

Table 38. Password transaction monitor properties

enrole.passwordtransactionmonitor.heartbeat	
	Specifies how often [in hours] the password transaction monitor checks for expired password transactions. Example (default): `enrole.passwordtransactionmonitor.heartbeat=1`

XML and DTD properties

XML and DTD properties are no longer used.

These properties are no longer used.

Table 39. XML and DTD properties

enrole.dtd.uri	
	Not used.

LDAP connection pool properties

LDAP connection pool properties are used to configure cache connection requests to the directory server.

Table 40 defines the properties used to configure the values that affect cache connection requests to the IBM Security Identity Manager directory server.

Table 40. LDAP connection pool properties

enrole.connectionpool.incrementcount	
	This property key and value affect performance tuning for IBM Security Identity Manager. They must be changed only by a qualified administrator. Specifies the number of connections that are created any time the LDAP connection pool is incremented to accommodate an increasing demand. Example (default): `enrole.connectionpool.incrementcount=3`
enrole.connectionpool.authentication	
	This property key and value affect performance tuning for IBM Security Identity Manager. They must be changed only by a qualified administrator. Specifies a list of space-separated authentication types of connections that can be pooled. Valid types are: • **none** - No authentication is required. • **simple** • **DIGEST-MD5** - Example (default): `enrole.connectionpool.authentication=none simple`
enrole.connectionpool.debug	

Table 40. LDAP connection pool properties (continued)

	This property key and value specify the level of debug output. Valid values are "fine" (trace connection creation and removal) and "all" (all debugging information). Valid values are: • **fine** - Trace connection creation and removal. • **all** - All debugging information. Example (default, commented out): `#enrole.connectionpool.debug=fine`
`enrole.connectionpool.initialpoolsize`	
	This property key and value affect performance tuning for IBM Security Identity Manager. They must be changed only by a qualified administrator. Specifies the initial number of physical LDAP connections to create for the LDAP connection pool. This value must be less than or equal to the value of the `maxpoolsize` property. Example (default): `enrole.connectionpool.initialpoolsize=50`
`enrole.connectionpool.maxpoolsize`	
	This property key and value affect performance tuning for IBM Security Identity Manager. They must be changed only by a qualified administrator. Specifies the maximum number of physical LDAP connections that can be created. Example (default): `enrole.connectionpool.maxpoolsize=100`
`enrole.connectionpool.prefsize`	
	This property key and value affect performance tuning for IBM Security Identity Manager. They must be changed only by a qualified administrator. Specifies the preferred number of physical LDAP connections that must be maintained concurrently. This number includes both in-use and idle connections. A size of zero or no value means that there is no preferred size. In that case, a request for a pooled connection results in a newly created connection if no idle ones are available. Example (no value): `enrole.connectionpool.prefsize=`
`enrole.connectionpool.protocol`	

Table 40. LDAP connection pool properties (continued)

	This property key and value affect performance tuning for IBM Security Identity Manager. They must be changed only by a qualified administrator. Specifies a list of space-separated protocol types of connections that can be pooled. Valid values are: • **plain** • **ssl** • **plain ssl** Example (default): `enrole.connectionpool.protocol=plain ssl`
`enrole.connectionpool.timeout`	
	This property key and value affect performance tuning for IBM Security Identity Manager. They must be changed only by a qualified administrator. Specifies the number of milliseconds that an idle connection can remain in the pool without being closed and removed from the pool. Example (default, commented out): `#enrole.connectionpool.timeout=10000`

Password encryption properties

Password encryption properties are used to configure password encryption.

Table 41 defines the properties used to configure password encryption.

Table 41. Encryption properties

`enrole.encryption.algorithm`	
	Do not modify this property key and value. Specifies the cipher suite to use for encryption. For example, `AES`. Example (default): `enrole.encryption.algorithm=AES`
`enrole.encryption.password`	
	Do not modify this property key and value. This value is specified during IBM Security Identity Manager installation. The value of the `enrole.encryption.password` property is moved into the `encryptionKey` property file. The value is encoded by default and is stored in the `encryptionKey` property file. Specifies the keystore password, in encrypted format, when AES is the encryption algorithm. For non-PBE based encryption algorithms (used for new IBM Tivoli® Identity Manager Version 5.0 installations), the password is used to encrypt the keystore that stores the private key. For more information about this property, see the `enrole.encryption.keystore` property. This value is specified during IBM Security Identity Manager installation.

Table 41. Encryption properties (continued)

`enrole.encryption.passwordDigest`	
	Do not modify this property key and value. Specifies the type of password digest used for an IBM Security Identity Manager password. Upgrading Tivoli Identity Manager from Version 4.6 continues to use the original hash algorithm until users change their passwords. This original algorithm is defined by the property `enrole.pre50.encryption.passwordDigest`. Valid values are: • **SHA-256** – Federal Information Processing Standards (FIPS)-approved hashing algorithm used by IBM Tivoli Identity Manager Version 5.0 for passwords. A random salt value is added to the data before it is hashed. • **SHA-384** – Federal Information Processing Standards (FIPS)-approved hashing algorithm, providing 384 bits of security (by truncating the output of the SHA-512 algorithm). A random salt value is added to the data before it is hashed. • **SHA-512** – Federal Information Processing Standards (FIPS)-approved hashing algorithm, providing 512 bits of security. A random salt value is added to the data before it is hashed. Example (default): `enrole.encryption.passwordDigest=SHA-256`
`enrole.pre50.encryption.passwordDigest`	
	Do not modify this property key and value. Upgrading IBM Tivoli Identity Manager from Version 4.6 adds this property dynamically to this properties file. Specifies the type of password digest used for IBM Security Identity Manager password data from IBM Tivoli Identity Manager versions before 5.0. The lack of a ":" in an encrypted IBM Security Identity Manager password value is used to identify such migrated data. **Note:** All new passwords, including changed migrated passwords, are stored with the `enrole.encryption.passwordDigest` algorithm. Example (default for migrated installations, not present for new installations): `enrole.pre50.encryption.passwordDigest=MD5`
`enrole.encryption.keystore`	
	Do not modify this property key and value. Specifies the keystore file name used to contain the randomly generated secret key for non-PBE based encryption algorithms, such as AES. This keystore file is protected with the `enrole.encryption.password` value. This file is in the *ISIM_HOME*`\data\keystore` directory. Example (default): `enrole.encryption.keystore=itimKeystore.jceks`

Challenge response encoding properties

Challenge response encoding properties determine whether a response is encoded as case sensitive or insensitive.

Table 42 on page 252 defines the properties used to encode a response as case sensitive or insensitive.

Table 42. Challenge response encoding properties

enrole.challengeresponse.responseConvertCase	
	Do not change this property key and value unless you are a qualified administrator. Specifies how CR responses are encoded before they are stored in the directory. Valid values are: • **lower** – Encode the CR as lowercase. • **upper** – Encode the CR as uppercase. • **none** – Do not encode the CR. Retain the case-sensitive response as is. Example (default): `enrole.challengeresponse.responseConvertCase=lower`

System listening port properties

System listening port properties are used to configure the listening port settings for the IBM Security Identity Manager Server.

Table 43 defines the properties used to configure the listening port settings for the IBM Security Identity Manager Server.

Table 43. System configuration properties

enrole.system.listenPort	
	Do not modify this property key and value. Specifies the TCP (non-secure communication) listening port value. This value is set during IBM Security Identity Manager installation. Example (default): `enrole.system.listenPort=80`
enrole.system.SSLlistenPort	
	Do not modify this property key and value. Specifies the Secure Sockets Layer (SSL) listening port value. This value is set during IBM Security Identity Manager installation. Example (default): `enrole.system.SSLlistenPort=443`

Mail properties

Mail properties are used to configure internal mail notification.

Table 44 defines the properties used to configure internal mail notification.

Table 44. Mail services properties

enrole.mail.notify

Table 44. Mail services properties (continued)

	Specifies whether the sending of workflow internal email is synchronized or not. Values include: • `SYNC` - Synchronized. • `ASYNC` - Asynchronized. Example (default): `enrole.mail.notify=ASYNC`

Workflow properties

Workflow properties are used to configure the core IBM Security Identity Manager workflow engine.

Table 45 defines the properties used to configure the core IBM Security Identity Manager workflow engine.

Table 45. Workflow configuration properties

Workflow configuration	
`enrole.workflow.lrucache.size`	
	Specifies the size of the cache used to temporarily use and access workflow objects. Do not change it unless directed by IBM support. Making this value too large can result in out of memory conditions oIBM Security Identity Manager Server. Example (default, commented out): `## enrole.workflow.lrucache.size=`*`number_of_entries`* where the default value of *number_of_entries* is 2000.
`enrole.workflow.notifyoption`	
	Do not change this property key and value unless you are a qualified administrator. Specifies the behavior of workflow email notifications. Values are: • `0 (NOTIFY_NONE)` – Security Identity Manager does not send email notifications when the workflow process completes. • `1 (NOTIFY_REQUESTER)` – A process completion notification is sent to the requester when the workflow process completes. Account email notifications are then sent to the requestee for the following account requests: New Account New Password Change Account Deprovision Account Suspend Account Restore Account For example, when the workflow process completes for a new account request, a process completion notification is sent to the requester. A new account notification is then sent to the requestee. Example (default): `enrole.workflow.notifyoption=1`

Table 45. Workflow configuration properties (continued)

`enrole.workflow.notifypassword`	
	Do not change this property key and value unless you are a qualified administrator. Specifies the type of email notification in a password transaction (caused when a user password is changed or automatically generated). Values are: • **true** – email notification of a password change can be sent to a user. The actual notification mechanism and whether to include the actual password in the email is dictated by the configuration of the `enrole.workflow.notification.newpassword` property value. • **false** – email notification of a password change is sent to a user. The email contains a URL where the user can obtain the password. The URL prompts the user for the shared secret. Example (default): `enrole.workflow.notifypassword=true`
`enrole.workflow.notifyaccountsonwarning`	
	Specifies whether account email notifications are sent when the account operation results in a warning. Values are: • **true** – Sends account email notifications. • **false** – Does not send account email notifications. Example (default): `enrole.workflow.notifyaccountsonwarning=false`
`enrole.workflow.maxretry`	
	Do not change this property key and value unless you are a qualified administrator. Specifies the number of times an attempt is made to start a workflow that initially failed. See also `enrole.workflow.retrydelay`. Example (default): `enrole.workflow.maxretry=2`
`enrole.workflow.retrydelay`	
	Do not change this property key and value unless you are a qualified administrator. Specifies the time delay [in milliseconds] between successive attempts to start a workflow application that initially failed. See also `enrole.workflow.maxretry`. Example (default): `enrole.workflow.retrydelay=60000`
`enrole.workflow.skipapprovalforrequester`	

Table 45. Workflow configuration properties (continued)

	Do not change this property key and value unless you are a qualified administrator. For a workflow activity that requires approval, this property specifies whether to skip the approval for other approvers if the requester is also an approver. Values are: • **true** – Skips approval for other approvers if the requester is also an approver. • **false** – Forces an approval check from other required approvers of the activity, *except* the requester (if the requester is also an approver). If the requester is a single approver as a result of participant resolution, then the approval is skipped even when value is set to `false`. Example (default): `enrole.workflow.skipapprovalforrequester=false`
`enrole.workflow.disablerequesteeapproval`	
	Do not change this property key and value unless you are a qualified administrator. For a workflow activity that requires approval, this property specifies whether to disable the requestee approval if the requestee is also an approver. Values are: • **true** – Disables the requestee approval if the requestee is also an approver. • **false** – Sends an approval check to the requestee and other resolved participants if the requestee is also an approver. The default value is `false`. Example (default): `enrole.workflow.disablerequesteeapproval=false` For more information, see *Planning* > *Workflow planning* > *Workflow participants* > *Disable requestee or requester approval* on the *IBM Security Identity Manager documentation*.
`enrole.workflow.disablerequesterapproval`	
	Do not change this property key and value unless you are a qualified administrator. IBM Security Identity Manager considers this property value only when the `enrole.workflow.skipapprovalforrequester` property value is set to `false`. For a workflow activity that requires approval, this property specifies whether to disable the requester approval if the requester is an approver. Values are: • **true** – A value set to `false` for the `enrole.workflow.skipapprovalforrequester` property disables automatic approval if the requester is a lone approver. • **false** – Works according to the value that you set for the `enrole.workflow.skipapprovalforrequester` property. Example (default): `enrole.workflow.disablerequesterapproval=false` For more information, see *Planning* > *Workflow planning* > *Workflow participants* > *Disable requestee or requester approval* on the *IBM Security Identity Manager documentation*.
`enrole.workflow.skipfornoncompliantaccount`	

Table 45. Workflow configuration properties (continued)

	Do not change this property key and value unless you are a qualified administrator. Specifies whether to engage the entitlement workflow that is associated with the account. Specifies when a system account modification is triggered as a result of a policy enforcement action. Values are: • **true** – Skips this action. • **false** – Does not skip this action. Example (default): `enrole.workflow.skipfornoncompliantaccount=true`
`enrole.workflow.distribution`	
	Do not change this property key and value unless you are a qualified administrator. Specifies whether workflow requests use the IBM Security Identity Manager shared queues, which allow for workload distribution. Values are: • **true** – Workflow requests are eligible for distribution. • **false** – Workflow requests are not eligible for distribution. Example (default): `enrole.workflow.distribution=true`
`enrole.workflow.async_completion_enabled`	
	Do not change this property key and value unless you are a qualified administrator. Specifies whether the system uses asynchronous completion checking for some system workflows, which can decrease database lock contention and improve performance. Values are: • **true** – Uses asynchronous completion checking. • **false** – Does not use asynchronous completion checking. Example (default): `enrole.workflow.async_completion_enabled=true`
`enrole.workflow.async_completion_interval_sec`	
	Do not change this property key and value unless you are a qualified administrator. Specifies the interval in seconds that the system checks to see whether certain system workflows are complete. Only applicable when `enrole.workflow.async_completion_enabled=true.` Example (default): `enrole.workflow.async_completion_interval_sec=30`
`enrole.workflow.notification.activitytimeout`	
	Do not change this property key and value unless you are a qualified administrator. Specifies the default Java class that generates the workflow activity timeout notification. Example (default, entered as a single line): `enrole.workflow.notification.activitytimeout= com.ibm.itim.workflow.notification.TemplateActivityTimeoutNotification`
`enrole.workflow.notification.processtimeout`	

Table 45. Workflow configuration properties (continued)

	Do not change this property key and value unless you are a qualified administrator. Specifies the default Java class that generates the workflow process timeout notification. Example (default, entered as a single line): `enrole.workflow.notification.processtimeout=com.ibm.itim.workflow.notification.TemplateProcessTimeoutNotification`
`enrole.workflow.notification.processcomplete`	
	Do not change this property key and value unless you are a qualified administrator. Specifies the default Java class that generates the notification for when a workflow process is completed. Example (default, entered as a single line): `enrole.workflow.notification.processcomplete=com.ibm.itim.workflow.notification.TemplateProcessCompleteNotification`
`enrole.workflow.notification.pendingwork`	
	Do not change this property key and value unless you are a qualified administrator. Specifies the default Java class that generates the notification for when a workflow process is completed for manual activities (Approvals and Requests for Information). Example (default, entered as a single line): `enrole.workflow.notification.pendingwork=com.ibm.itim.workflow.notification.TemplatePendingWorkNotification`
`enrole.workflow.notification.newaccount`	
	Do not change this property key and value unless you are a qualified administrator. Specifies the default Java class that generates the notification for when a workflow process is completed for a new account. Example (default, entered as a single line): `enrole.workflow.notification.newaccount=com.ibm.itim.workflow.notification.TemplateNewAccountNotification`
`enrole.workflow.notification.newpassword`	

Table 45. Workflow configuration properties (continued)

	Do not change this property key and value unless you are a qualified administrator.
	
	Specifies the default Java class that generates a notification when a user changes a password. This property is used only when the value for the property is `true`.
	`enrole.workflow.notifypassword=true`
	
	This property responds to the following three-password change scenarios:
	• When a user changes the password for the account
	• When the administrator forces a password change on the account
	• When a user is successfully identified through the password challenge/response feature, and challenge/response is configured.
	
	Valid classes include:
	• **NewPasswordNotification**
	Email notification that includes the password (in ASCII text) is sent to a user (default).
	• **EmptyNotificationFactory**
	Suppresses email notification. The preferred method for suppressing any notification is through the Workflow Notification GUI.
	• **PasswordChangeNotificationFactory**
	Email notification that does not include the password is sent to a user. Message body says: "Process completed".
	
	The `EmptyNotificationFactory` and `PasswordChangeNotificationFactory` classes are in the `examples.jar` package in the `examples` directory.
	
	Example (default, entered as a single line):
	`enrole.workflow.notification.newpassword=com.ibm.itim.workflow.notification.TemplateNewPasswordNotification`

`enrole.workflow.notification.deprovision`	
	Do not change this property key and value unless you are a qualified administrator. Specifies the default Java class that generates deprovisioning notification. Example (default, entered as a single line): `enrole.workflow.notification.deprovision=com.ibm.itim.workflow.notification.TemplateDeprovisionNotification`
`enrole.workflow.notification.workorder`	
	Do not change this property key and value unless you are a qualified administrator. Specifies the default Java class that generates work order notifications. Example (default, entered as a single line): `enrole.workflow.notification.workorder=com.ibm.itim.workflow.notification.TemplateWorkOrderNotification`
`enrole.workflow.notification.changeaccount`	

Table 45. Workflow configuration properties (continued)

	Do not change this property key and value unless you are a qualified administrator. Specifies the default Java class that generates account change notifications. Example (default, as a single line): `enrole.workflow.notification.changeaccount= com.ibm.itim.workflow.notification.TemplateChangeAccountNotification`
`enrole.workflow.notification.restoreaccount`	
	Do not change this property key and value unless you are a qualified administrator. Specifies the default Java class that generates account restoration notifications. Example (as a single line): `enrole.workflow.notification.restoreaccount= com.ibm.itim.workflow.notification.TempateRestoreAccountNotification`
`enrole.workflow.notification.suspendaccount`	
	Do not change this property key and value unless you are a qualified administrator. Specifies the default Java class that generates account suspension notifications. Example (as a single line): `enrole.workflow.notification.suspendaccount= com.ibm.itim.workflow.notification.TemplateSuspendAccountNotification`

Reconciliation properties

Reconciliation properties are used to configure the reconciliation process where data retrieved from agents is synchronized in the IBM Security Identity Manager database.

Table 46 defines the properties used to configure the values that affect the reconciliation process where data retrieved from agents is synchronized in the IBM Security Identity Manager database.

Table 46. Reconciliation properties

Reconciliation configuration	
`enrole.reconciliation.accountcachesize`	
	Do not change this property key and value unless you are a qualified administrator. Specifies the maximum size of the cache for existing accounts cache that is used for the reconciliation process. Setting a value larger than the default might cause processing of reconciliations to fail. Example (default): `enrole.reconciliation.accountcachesize=2000`
`enrole.reconciliation.threadcount`	

Table 46. Reconciliation properties (continued)

	Do not change this property key and value unless you are a qualified administrator. Specifies the number of threads that are used to handle reconciled entries. This number of threads is created for each reconciliation process. Example (default): `enrole.reconciliation.threadcount=8`
`enrole.reconciliation.failurethreshold`	
	Do not change this property key and value unless you are a qualified administrator. Specifies the maximum number of local accounts to delete at the end of reconciliation. If the value is exceeded, then no local account or supporting data entries are deleted. If the value is followed by a percent sign (%), specifies the maximum as percentage compared with total of (local accounts at reconciliation start plus the new accounts returned by reconciliation). A value of 100% specifies that there is no limit. Example (default, commented out): `#enrole.reconciliation.failurethreshold=100%`
`enrole.reconciliation.logTimeInterval`	
	Do not change this property key and value unless you are a qualified administrator. Specifies the time interval in seconds for reconciliation progress trace log messages. A value of zero disables this time interval. Example (default, commented out): `#enrole.reconciliation.logTimeInterval=600`
`enrole.reconciliation.logEveryNResults`	
	Do not change this property key and value unless you are a qualified administrator. Specifies the count for reconciliation progress trace log messages. A value of zero disables this count. Example (default, commented out): `#enrole.reconciliation.logEveryNResults=5000`
Unsolicited notification events	
`account.EventProcessorFactory`	
	Do not modify this property key and value. Specifies the built-in Java class for the account event processor factory. Example (default, entered as a single line): `account.EventProcessorFactory=com.ibm.itim.remoteservices.ejb.reconciliation.AccountEventProcessorFactory`
`person.EventProcessorFactory`	

Table 46. Reconciliation properties (continued)

	Do not modify this property key and value. Specifies the built-in Java class for the person event processor factory. Example (default, entered as a single line): `person.EventProcessorFactory=com.ibm.itim.remoteservices.ejb.reconciliation.PersonEventProcessorFactory`
Reconciliation processing	
`account.ReconEntryHandlerFactory`	
	Do not modify this property key and value. Specifies the built-in Java class for the account entry handler factory. Example (default, entered as a single line): `account.ReconEntryHandlerFactory=com.ibm.itim.remoteservices.ejb.mediation.AccountEntryHandlerFactory`
`person.ReconEntryHandlerFactory`	
	Do not modify this property key and value. Specifies the built-in Java class for the person entry handler factory. Example (default, entered as a single line): `person.ReconEntryHandlerFactory=com.ibm.itim.remoteservices.ejb.mediation.PersonEntryHandlerFactory`
`enrole.reconciliation.accountChangeFormatter`	
	Do not change this property key and value unless you are a qualified administrator. When specified, this property allows you to customize how local attribute changes that are detected during reconciliation are formatted and stored. The default behavior can be overridden by specifying the fully qualified Java class name of an alternative implementation. Example (assuming Java class `com.example.custom.AccountChangeFormatter` is a custom implementation of interface `com.ibm.itim.remoteservices.ejb.mediation.IAccountChangeFormatter`). The example is entered as a single line: `enrole.reconciliation.accountChangeFormatter=com.example.custom.AccountChangeFormatter`
Deferring requests for failed remote resources	
`com.ibm.itim.remoteservices.ResourceProperties.DEFER_FAILED_RESOURCE`	

Table 46. Reconciliation properties (continued)

	Do not modify this property key and value. Specifies whether to defer requests to failed resources and wait for resource to restart before it sends them. Valid values are: • **true** – Defers requests to failed resources and waits for the resource to restart. • **false** – If the resource fails, requests follows the configured workflow retry mechanism before it terminates as failed. See `enrole.workflow.maxretry` and `enrole.workflow.retrydelay`. Example (default): `com.ibm.itim.remoteservices.ResourceProperties.DEFER_FAILED_RESOURCE=true`
`remoteservices.remotepending.interval`	
	Do not modify this property key and value. Specifies the interval in seconds (120 minimum to 3600 maximum) to check whether failed resources restart. Example (default): `remoteservices.remotepending.interval=600`
`com.ibm.itim.remoteservices.ResourceProperties.MAX_REQUEST_TIME`	
	Do not modify this property key and value. Specifies the maximum time in seconds that a request to a resource can be outstanding. It includes time in pending state for asynchronous requests, or deferred requests due to a service failure or request backlog. Valid values are: • **-1** – Unlimited • **60 + (value of remoteservices.remotepending.interval)** – Minimum time interval for outstanding requests. Example (default): `com.ibm.itim.remoteservices.ResourceProperties.MAX_REQUEST_TIME=-1`
`remoteservices.remotepending.restart.retry`	
	Do not modify this property key and value. Specifies the time interval in minutes that pending requests generated from the restart of a failed service are given to complete. When the time interval ends, the server retries the requests. Example (default): `remoteservices.remotepending.restart.retry=1440`
`com.ibm.itim.remoteservices.DSML2ServiceProvider.modifyAsREPLACE`	

Table 46. Reconciliation properties (continued)

	Do not change this property key and value unless you are a qualified administrator. For remote services, specifies the DSMLv2 (deprecated) provider mode of sending a modify request for attributes. Values include: • **true** – Use the `REPLACE` operation. • **false** – Use the `ADD` and `DELETE` operations. Example (default): `com.ibm.itim.remoteservices.DSML2ServiceProvider.modifyAsREPLACE=true`

Shared secret properties

Shared secret properties are used to configure the level of protection of the shared secret code.

Table 47 defines the properties used to configure the level of protection of the shared secret code.

The shared secret is used by an account owner to retrieve a new or changed password for an account when the system is configured to not email passwords in the clear (that is, the value of `enrole.workflow.notifypassword=false`). This property determines whether the stored shared secret is hashed for additional protection.

Table 47. Shared secret hashing properties

`enrole.sharedsecret.hashed`	
	Do not change this property key and value unless you are a qualified administrator. Specifies whether the shared secret code is hashed (secure) or not hashed (not secure). Values include: • **true** – Store the shared secret as hashed. • **false** – Store the shared secret as not hashed. Example (default): `enrole.sharedsecret.hashed=false`

Lifecycle rule properties

Lifecycle rule properties define values such as the partition size used for lifecycle rules.

Table 48 defines the properties used to configure lifecycle rules.

Table 48. Lifecycle rule properties

`enrole.lifecyclerule.partition.size`

Table 48. Lifecycle rule properties (continued)

	Do not change this value unless requested by IBM support. Specifies the size of the data partitions for processing lifecycle rules. This parameter determines how much data is processed in a single step. Example (default): `enrole.lifecyclerule.partition.size=100`

Product name properties

Product name properties identify this product.

Table 49 defines the property used to identify the product.

Table 49. Product property

`enrole.product.name`	
	Do not change this name. This property key identifies the product name as IBM Security Identity Manager. Example (default): `enrole.product.name=ITIM Enterprise`

Application client request properties

Application client request properties define the properties used to configure the lifetime, or timeout, value for the authentication token used to allow third-party communication with IBM Security Identity Manager Server.

Table 50 defines the properties used to configure the lifetime, or timeout, value for the authentication token used by the IBM Security Identity Manager application API to allow third-party applications to communicate with the IBM Security Identity Manager Server.

Table 50. Application client request properties

`authTokenTimeout`	
	Specifies timeout value in hours for the authentication token that is used for communication between third-party applications (with the IBM Security Identity Manager application API) and the IBM Security Identity Manager Server. A value of -1 indicates that there is no timeout for the authentication token. Example (default): `authTokenTimeout=48`

Reverse password synchronization properties

Reverse password synchronization properties are used to configure reverse password synchronization.

Table 51 on page 265 defines the properties used to configure reverse password synchronization.

Table 51. Reverse password synchronization properties

`reversePasswordSynch.bypassPwdValidationOnOrphanAccount`	
	Specifies whether to bypass the password validation on the orphan account when the request is submitted from the agent. Valid values are: • **true** – Bypass password validation. • **false** – Validate passwords. Example (default): `reversePasswordSynch.bypassPwdValidationOnOrphanAccount=false`
`enrole.passwordsynch.module.sendMail`	
	Specifies whether to enable or disable email notifications when password synchronization is triggered by the reverse password synchronization agent, not from the IBM Security Identity Manager graphical user interface. Valid values are: • **true** – Enable email notifications. • **false** – Disable email notifications. Example (default): `enrole.passwordsynch.module.sendMail=false`

Post office properties

Post office properties are used to configure the post office for email collection.

Table 52 defines the properties for testing post office configuration.

Table 52. Post office properties

`enrole.postoffice.test.subject1`
`enrole.postoffice.test.textbody1`
`enrole.postoffice.test.xhtmlbody1`	

	Specifies the contents of the emails that are used when you test the post office configuration. It is one of three emails to which the template is applied.
	Example (default):
	`enrole.postoffice.test.subject1=This is subject 1`
	`enrole.postoffice.test.textbody1=This is the text body 1`
	`enrole.postoffice.test.xhtmlbody1=This is the xhtml body 1`

`enrole.postoffice.test.subject2`
`enrole.postoffice.test.textbody2`
`enrole.postoffice.test.xhtmlbody2`	

	Specifies the contents of the emails that are used when you test the post office configuration. It is one of three emails to which the template is applied.
	Example (default):
	`enrole.postoffice.test.subject2=This is subject 2`
	`enrole.postoffice.test.textbody2=This is the text body 2`
	`enrole.postoffice.test.xhtmlbody2=This is the xhtml body 2`

`enrole.postoffice.test.subject3`
`enrole.postoffice.test.textbody3`
`enrole.postoffice.test.xhtmlbody3`	

Table 52. Post office properties (continued)

	Specifies the contents of the emails that are used when you test the post office configuration. It is one of three emails to which the template is applied.
	Example (default):
	`enrole.postoffice.test.subject3=This is subject 3`
	`enrole.postoffice.test.textbody3=This is the text body 3`
	`enrole.postoffice.test.xhtmlbody3=This is the xhtml body 3`

`enrole.postoffice.test.topic`	
	Specifies the topic of the email that is used when you test the post office configuration. The three test emails, whose content is defined by the preceding properties, all have this topic. The post office function gathers and stores emails by topic and locale, It then aggregates and sends them as one email on a configured interval, such as once a day or once a week. This method prevents flooding the recipient with many individual emails for a type of event. The topic data usually indicates the type of event. It is also made available to the programming environment that is activated when the gathered emails are aggregated into one summarizing email. In this way, the topic under which all of these emails were gathered can be prominently displayed in the aggregate email that is sent. Example (default): `enrole.postoffice.test.topic=topic1`
`enrole.postoffice.test.locale`	
	Specifies the locale for the language that is used in an email. Example (default): `enrole.postoffice.test.locale=en_US`

Database resource bundle properties

Database resource bundle properties determine the refresh interval for the database resource bundle.

Table 53 defines the properties used to determine the refresh interval for the database resource bundle.

Table 53. Database resource bundle properties

`enrole.databaseresourcebundle.refreshInterval`	
	Specifies how many minutes to wait before `DatabaseResourceBundle` is checked for changes and reloaded. Example (default): `enrole.databaseresourcebundle.refreshInterval=5`

Database cleanup properties

Database cleanup properties define the parameters to clean up session information in the database.

Table 54 defines the parameters for the policy analysis scavenger thread to clean up session information in the database.

Table 54. Database cleanup properties

`provisioning.policy.preview.cleanup.interval`	
	Specifies the interval in minutes that the scavenger thread scans the database. Example: `provisioning.policy.preview.cleanup.interval=30`
`provisioning.policy.analysis.idle.timeout`	
	Represents the expired time setting for a policy analysis session. The scavenger thread cleans up the staged data of a policy analysis session if the session ends at an interval that is greater than the timeout value. The timeout value might be 120 minutes. Example: `provisioning.policy.analysis.idle.timeout=120`

Create password check box properties

Create password check box properties define the default check box properties to create a password.

Table 55 defines the default create password check box properties.

Table 55. Create password check box default properties

`enrole.CreatePassword`	
	Specifies whether a password is created automatically. Valid values are: • **true** – Create a password. • **false** – Do not create a password. The user must type in the password. Example (default): `enrole.CreatePassword=true`

Access catalog properties

The `com.ibm.itim.accesscatalog.groupIntersectionJoin.enabled` enables support for searching group access when requesting access in the Identity Service Center when `Intersection Join directive` is used for the group attribute. The `com.ibm.itim.accesscatalog.customJoin.enabled` enables support for searching group access when requesting access in the Identity Service Center when `Custom Join directive` is used for the group attribute.

Table 56. Access catalog properties

`com.ibm.itim.accesscatalog.groupIntersectionJoin.enabled`

Table 56. Access catalog properties (continued)

	Do not change this property value unless you are a qualified administrator. Enables support for searching group access when requesting access in the Identity Service Center in the case where `Intersection Join directive` is used for the group attribute. Values include: • **true** • **false** The default is `false`. Example (default): `com.ibm.itim.accesscatalog.groupIntersectionJoin.enabled=false`
`com.ibm.itim.accesscatalog.customJoin.enabled`	
	Do not change this property value unless you are a qualified administrator. Enables support for searching group access when requesting access in the Identity Service Center in the case where `Custom Join directive` is used for the group attribute. Values include: • **true** • **false** The default is `false`. Example (default): `com.ibm.itim.accesscatalog.customJoin.enabled=false`

Identity feed properties

Identity feed properties define a default identity feed action, such as whether to suspend an account.

Table 57 defines the default identity feed properties.

Table 57. Default identity feed properties

`enrole.suspend.accounts.identity.feed`	
	Specifies whether all of a user's accounts are suspended when the person is suspended during an identity feed. Valid values are: • **true** – Suspend all accounts of a suspended user. • **false** – Do not suspend all accounts of a suspended user. Example (default): `enrole.suspend.accounts.identity.feed=true`

Upgrade properties

Upgrade properties define values for the upgrade of a specific release of IBM Security Identity Manager.

Table 58 defines the product upgrade properties.

Table 58. Default upgrade properties

`minUpgradeVersion`	
	Specifies the minimum version that the upgrade supports for a specific release of IBM Security Identity Manager. Example (default): `minUpgradeVersion=5.1`
`file.merge.list`	

	Specifies which properties files are merged during the upgrade of IBM Security Identity Manager.
	
	Example (default):
	`file.merge.list=enRole \`
	`enRoleLDAPConnection \`
	`enRoleDatabase \`
	`enRoleLogging \`
	`enRoleMail \`
	`ui \`
	`CustomLabels \`
	`CustomLabels_en \`
	`enRoleAuthentication \`
	`adhocreporting \`
	`enroleworkflow \`
	`enroleAuditing \`
	`SelfServiceScreenText \`
	`SelfServiceScreenText_en \`
	`SelfServiceHelp \`
	`SelfServiceUI \`
	`SelfServiceHomePage\`
	`scriptframework\`
	`encryptionKey\`
	`KMIPServer`

Multiple password-synch agent properties

Multiple password-synch agent properties are used to configure the IBM Security Identity Manager Server to support multiple password-synchronization agents.

Table 59 defines the properties used to configure the support for multiple password-synch agents.

Table 59. Multiple password-synch agent properties

`enrole.passwordsynch.enabledonresource`

Table 59. Multiple password-synch agent properties (continued)

	Specifies whether to enable or disable the support for multiple password-synch agents. Valid values are: • **true** – Enable the support for multiple password-synch agents • **false** – Disable the support for multiple password-synch agents Example (default): `enrole.passwordsynch.enabledonresource=false`
`enrole.passwordsynch.toleranceperiod`	
	Specifies the maximum time duration, in *seconds*, between a password change request sent from the IBM Security Identity Manager Server to the password synch resource, and receiving a reverse password synch request from the plug-in installed on the password synch resource. Example (default): `enrole.passwordsynch.toleranceperiod=60`
`enrole.PasswordSynchStoreMonitor.heartbeat`	
	Specifies the password synch transaction monitor heartbeat, in *hours*. Example (default): `enrole.PasswordSynchStoreMonitor.heartbeat=1`

Concurrency properties

Account concurrency properties determine how to resolve multiple provisioning requests for the same account ID.

Table 60. Account concurrency properties

`account.provision.concurrency.resolution`
Specifies which conflict resolution method is used when a concurrency issue occurs. Select from the following values: • 0 - Change the concurrent account add operations to account modify operations. • 1 - Add the account with a newly generated account user ID • 2 - No operation override. Fail the account provisioning. Example (default): `account.provision.concurrency.resolution=0`

Required field properties

These properties are used to configure whether fields in the user interface are required to be completed by the user.

Table 61 defines the properties that are used to determine whether a field in the user interface is a required field.

Table 61. Required field properties

`enrole.justificationRequired`

Table 61. Required field properties (continued)

	Specifies whether the **Justification** field is a required field. By default, the **Justification** field is not displayed in the user interface. Setting this property to `true` causes the **Justification** property to be displayed. It also sets the field as required to be completed by the user. Example (default): `enrole.justificationRequired=false`

Chapter 18. Virtual appliance command line interface

Access the command line interface (CLI) of the virtual appliance by using either an `ssh` session or the console.

1. From the command-line interface, log on the virtual appliance. The following message is displayed:

```
Welcome to the IBM Security Identity Manager appliance
Enter "help" for a list of available commands
```

2. To see a list of available commands, enter **`help`** at the command-line prompt. The **`help`** command provides detailed information about each command from the list.

The following example shows an `ssh` session that is used to access the virtual appliance.

```
usernameA@example.com> ssh -l admin isimva.example.com
admin@isimva.example.com's password:
Welcome to the IBM Security Identity Manager appliance
Enter "help" for a list of available commands
isimva.example.com> help
Current mode commands:
fips                View FIPS 140-2 state and events.
firmware            Work with firmware images.
fixpacks            Work with fix packs.
isim                Work with the IBM Security Identity Manager settings.
license             Work with licenses.
lmi                 Work with the local management interface.
management          Work with management settings.
snapshots           Work with policy snapshot files.
support             Work with support information files.
tools               Work with network diagnostic tools.
Global commands:
back                Return to the previous command mode.
exit                Log off from the appliance.
help                Display information for using the specified command.
reboot              Reboot the appliance.
shutdown            End system operation and turn off the power.
top                 Return to the top level.
isimva.example.com:isim>
```

You can also access the console by using the appropriate VMware software. For example, VMware vSphere Client.

Note: The CLI contains only a subset of the function available from the graphical user interface.

Global commands

The following list describes a high-level overview of the global functions that are available in the command line interface.

`back` Returns to the previous command mode.

`exit` Logs off from the appliance.

`help` Displays information for using the specified command.

`reboot` Restarts the appliance.

shutdown
Ends system operations and turns off the power.

top Returns to the top level.

Current® mode commands

fips View FIPS 140-2 state and events. FIPS commands are available only if FIPS mode is enabled.

firmware
Work with firmware images.

fixpacks
Work with fix packs.

isim Work with the IBM Security Identity Manager settings. See Chapter 19, "IBM Security Identity Manager(isim) commands," on page 279.

license
Work with licenses.

lmi Work with the local management interface.

management
Work with management settings.

snapshots
Work with policy snapshot files.

support
Work with support information files.

tools Work with network diagnostic tools.

fips commands

The **fips** command provides the option to view FIPS 140-2 state and events.

Usage

CLI:

```
isimvasvr > fips
isimvasvr: fips >
```

Note: The **fips** commands are available only if FIPS mode is enabled for the IBM Security Identity Manager virtual appliance.

Federal Information Processing Standards (FIPS) are guidelines that are set for software and hardware computer security products.

status Displays the status of FIPS 140-2 mode.

If FIPS mode is enabled successfully on the virtual appliance, the following message is displayed.

```
FIPS 140-2 Status: OK
Appliance has enabled FIPS mode successfully.
```

If the virtual appliance is in an error state, the following message is displayed:

```
FIPS 140-2 Status: Error
Appliance has entered FIPS error state.
```

view_log
Displays the FIPS 140-2 messages in the system log.

firmware commands

Use the firmware commands to work with firmware images.

Usage

CLI:

```
isimvasvr > firmware
isimvasvr: firmware >
```

backup Backs up firmware on the active partition to the inactive partition.

get_comment
Shows the comment that is associated with a firmware image.

get_info
Shows the version information that is associated with a firmware image.

list Lists information about the installed firmware images. Firmware information includes the active firmware image, a description of the firmware, the date the firmware was installed and optional backup information.

set_comment
Replaces the comment that is associated with a firmware image.

swap_active
Swaps the active firmware image. The appliance restarts the system with the inactive firmware image.

fixpacks commands

The **fixpacks** command provides options to work with the fix packs. The corresponding task can be completed by using the graphical user interface. Go to **Manage** > **Updates and Licensing** > **Fix Packs**.

Usage

CLI:

```
isimvasvr > fixpacks
isimvasvr: fixpacks >
```

install
Installs the available fix packs on the inserted USB device.

list Lists the available fix packs on the inserted USB device.

rollback
Uninstalls the most recently installed fix pack.

view_history
Shows the installation history for all fix packs.

license commands

The **license** command provides options to work with the licenses.

Usage

CLI:

```
isimvasvr > license
isimvasvr: license >
```

install
Installs a license file from an inserted USB device.

list Lists the available license files on the inserted USB device.

show Displays the current active license information.

lmi commands

The **lmi** command provides options to work with the local management interface.

Usage

CLI:

```
isimvasvr > lmi
isimvasvr: lmi >
```

reset_lmi_cert
Resets the server certificate for the local management interface to a self-signed certificate.

restart
Restarts the local management interface.

security
Provides options to work with security settings for the local management interface.

trace Provides options to work with the trace settings for the local management interface.

management command

The **management** command provides options to work with management settings.

Usage

CLI:

```
isimvasvr > management
isimvasvr: management >
```

dns Provides options to work with the virtual appliance DNS settings.

hostname
Provides options to work with the virtual appliance host name.

interfaces
Provides options to work with the management interface settings.

set_password
Sets the virtual appliance password.

snapshots command

The snapshots command provides options to work with the snapshots. You can also complete the corresponding task by using the graphical user interface. Go to **Manage** > **System Settings** > **Snapshots**.

Usage

CLI:

```
isimvasvr > snapshots
isimvasvr: snapshots >
```

apply Applies a policy snapshot file to the system.

Note: You must restart the virtual appliance after you apply the snapshot.

create Creates a snapshot of current policy files.

delete Deletes a policy snapshot file.

download
Downloads a policy snapshot file to a USB flash drive.

get_comment
Shows the comment that is associated with a policy snapshot file.

list Lists the policy snapshot files.

set_comment
Replaces the comment that is associated with a policy snapshot file.

upload Uploads a policy snapshot file from a USB flash drive.

support command

The function generates the support files. The corresponding task can be completed by using the graphical user interface. Go to **Manage** > **System Settings** > **Support Files**.

Usage

CLI:

```
isimvasvr > support
isimvasvr: support >
```

create Creates a support information file.

delete Deletes a support information file.

download
Downloads a support information file to a USB flash drive.

get_comment
Shows the comment that is associated with a support information file.

list Lists the support information files.

set_comment
Replaces the comment that is associated with a support information file.

tools command

The tools command provides options to work with network diagnostic tools.

Usage

CLI:

```
isimvasvr > tools
isimvasvr: tools >
```

connect
Tests the network connection to a certain port on a specified host.

connections
Displays the network connections for the appliance.

nslookup
Queries internet domain name servers.

ping Sends an ICMP ECHO_REQUEST to network hosts.

traceroute
Traces a packet from a computer to a remote destination. Shows the required number of hops for a packet that is required to reach the destination and the duration of each hop.

Chapter 19. IBM Security Identity Manager(isim) commands

The initial virtual appliance settings wizard runs the first time that an Administrator logs on to the command line interface (CLI) of an unconfigured virtual appliance. The topic provides information about the sub sections of the virtual appliance CLI command that is specific to IBM Security Identity Manager.

Usage

CLI:

```
isimvasvr > isim
isimvasvr: isim >
```

Use the virtual appliance command-line interface to run IBM Security Identity Manager commands.

1. Access the command-line interface of the virtual appliance by using either a **ssh** session or the console.
2. From the command-line interface, log on the virtual appliance. The following message is displayed:

   ```
   Welcome to the IBM Security Identity Manager appliance
   Enter "help" for a list of available commands
   ```

3. Enter **isim** to use the IBM Security Identity Manager commands.

   ```
   isimvasrv > isim
   ```

4. Optional: To see a list of available commands, enter the **help** command.

   ```
   isimvasrv : isim > help
   ```

The commands are categorized into the following categories:

```
jvm_heapsize             Change the jvm heap size for ISIM DMGR, Application and Message Member.
jvm_property             Work with the Application Server JVM properties.
keystore_password        Work with the ISIM keystore.
langpack                 Work with the ISIM Server language packs.
logs                     Work with the IBM Security Identity Manager log files.
migration                Work with the ISIM migration utilities.
nodes_administration     Work with ISIM application server nodes in the cluster.
single_sign_on_settings  Work with single sign-on.
upgrade                  Work with the IBM Security Identity Manager upgrade.
utilities                Work with the IBM Security Identity Manager utilities.
```

Learn more about available commands by entering **help** on any of the subcommands.

jvm_heapsize command

You can view and change the maximum heap size of the Java™ virtual machine (JVM) for the Security Identity Manager WebSphere® Application Server profiles.

The JVM has heap memory, which is a runtime data area where memory for all class instances and arrays are allocated. Heap memory for either servers are created at JVM start-up with a default value that is based on the size of RAM.

If the default value is insufficient to suit the memory consumption of either servers, you can change the maximum heap size to a value that you prefer.

After you change the maximum heap size for either or all server profiles, restart the IBM Security Identity Manager and Cluster Manager server to apply your changes.

Usage

```
isimvasvr > isim
isimvasvr: isim > jvm_heapsize
isimvasvr: jvm_heapsize >
```

reset_max_heapsize
Reset maximum heap size for ISIM DMGR, Application and Message Member to default value.

set_max_heapsize
Set maximum heap size for ISIM DMGR, Application and Message Member.

show_max_heapsize
Show maximum heap size for ISIM DMGR, Application and Message Member.

Examples

- At the isimvasrv:jvm_heapsize prompt, enter **show_max_heapsize**.

```
1: DMGR
2: APP_Member
3: MSG_Member Enter index: 1
DMGR max heap size(MB): 2048
```

- Set a maximum heap size.

  At the isimvasrv:jvm_heapsize prompt, enter **set_max_heapsize**.

```
1: DMGR
2: APP_Member
3: MSG_Member
Enter index: 1
Enter max jvm heap size(MB): 5000
Updating max jvm heap size for DMGR.
Please wait...
Max heap size updated successfully.
Restart Cluster Manager Server and IBM Security Identity Manager server.
```

- Reset the maximum heap size.

  At the isimvasrv:jvm_heapsize prompt, enter **reset_max_heapsize**.

```
1: DMGR
2: APP_Member
3: MSG_Member
Enter index: 1
Are you sure you want to reset the jvm maximum heap size?
Enter YES to confirm that you want to continue: YES
Reseting max jvm heap size for DMGR. Please wait...
Max heap size updated successfully.
```

Important: Restart the Cluster Manager Server and IBM Security Identity Manager Server to apply the new settings.

jvm_property command

The **jvm_property** command provides options to work with the Application Server JVM properties.

Usage

CLI:

```
isimvasvr > isim
isimvasvr: isim > jvm_property
isimvasvr: jvm_property >
```

add Adds a JVM property in the application server.

delete Deletes an existing JVM property from the application server.

update Updates an existing JVM property in the application server.

Examples

- Add a JVM property to the application server.

  At the *isimvasvr*: jvm_property prompt, enter **add**.

```
Property name : com.ibm.websphere.webservices.soap.enable.legacy.get.behavior
Property value : true
Adding JVM property JVM Property added successfully.
Restart Identity Manager server to apply the new settings.
```

- Update a JVM property.

  At the *isimvasvr*: jvm_property prompt, enter **update**.

```
Existing JVM Properties :
 1: com.ibm.websphere.webservices.soap.enable.legacy.get.behavior=true
Enter index: 1
Property value : false
Updating JVM property
JVM Property updated successfully.
Restart IBM Security Identity Manager to apply the new settings.
```

- Delete a JVM property.

  At the *isimvasvr*: jvm_property prompt, enter **delete**.

```
Existing JVM Properties :
 1: com.ibm.websphere.webservices.soap.enable.legacy.get.behavior=false
Enter index: 1
This operation deletes an existing JVM property from Application server.
Selected property : com.ibm.websphere.webservices.soap.enable.legacy.get.behavior=false
Do you wish to delete this property?
Enter 'YES' to confirm: YES
Deleting JVM property
JVM Property deleted successfully.
Restart IBM Security Identity Manager to apply the new settings.
```

keystore_password command

The **keystore_password** command provides options to work with the IBM Security Identity Manager keystore.

Usage

CLI:

```
isimvasvr > isim
isimvasvr: isim > keystore_password
isimvasvr: keystore_password >
```

update Updates the IBM Security Identity Manager password in the IBM Security Identity Manager encryption properties.

Example

Update the IBM Security Identity Manager keystore password.

At the *isimvasvr*: keystore_password prompt, enter **update**.

```
Enter password:
Confirm password:
Successfully set keystore password. Restart the Identity Manager server.
Successfully updated the ISIM keystore password.
```

Important: Restart the IBM Security Identity Manager server to apply the new settings.

langpack command

The **langpack** command provides options to work with the IBM Security Identity Manager language packs.

Usage

CLI:

```
isimvasvr > isim
isimvasvr: isim > langpack
isimvasvr: langpack >
```

install
Installs the language pack in the IBM Security Identity Manager server.

list Lists the installed language packs in the IBM Security Identity Manager server.

Examples

- Install a language pack.

  At *isimvasvr*: langpack prompt, enter **install**.

```
Select the language to install :
1: French
2: German
3: Italian
4: Czech
5: Greek
6: Hungarian
7: Russian
8: Hebrew
9: Portuguese
10: Polish
11: Spanish
12: Arabic
13: Japanese
14: Korean
15: Simplified Chinese
16: Traditional Chinese
Enter index: 3
Installing language pack for "Italian" Language pack installed successfully.
```

- List installed languages packs.

  At the *isimvasvr*: langpack prompt, enter **list**.

```
Installed languages on ISIM server
English
Italian
```

logs command

The **logs** command provides options to work with the IBM Security Identity Manager log files.

Usage

CLI:

```
isimvasvr > isim
isimvasvr: isim > logs
isimvasvr: logs >
```

clear Clears the log files on the system.

clear_ffdc
Clears all the FFDC log files on the system.

clear_tranlog
Clears all transaction logs on the system.

monitor
Provides options to monitor the log files on the system.

Examples

- Clear log files.

  At the *isimvasvr*: logs enter **clear**.

```
Options:
1: LMI
2: Cluster Manager
3: Application Server
4: Message Server
5: ISIM Server
6: SDI
Enter index: 1
Options:
1: console.log
2: messages.log
3: trace.log
Enter index: 1
```

- Clear all the FFDC log files on the system.

  At the *isimvasvr*: logs prompt, enter **clear_ffdc**

```
This operation clears all the FFDC log files from the system. You can back up your FFDC logs
using support package.
Do you wish to clear all the FFDC logs?.
Enter 'YES' to confirm: YES

All the FFDC log files has been cleared.
```

Tailing logs and archiving logs

You can generate tailing logs and archiving logs through the command-line interface in the IBM Security Identity Manager virtual appliance.

About this task

To see a list of available commands, enter the `help` command at the command-line prompt. The **help** command provides detailed information about each command from the list.

Procedure

1. From the command-line interface, log on to the IBM Security Identity Manager virtual appliance. The following message is displayed:

   ```
   Welcome to the IBM Security Identity Manager appliance
   Enter "help" for a list of available commands
   ```

2. Enter the `help` command at the `isimvasrv` prompt for a list of available commands.
3. Enter the `isim` command at the `isimvasrv` prompt.
4. Enter the `help` command at the `isimvasrv:isim` prompt for a list of available commands.
5. Enter the `logs` command at the `isimvasrv:isim` prompt.
6. Enter the `help` command at the `isimvasrv:isim:logs` prompt for a list of available commands. The following result is displayed:

   ```
   Current mode commands:
   clear               Clear the log files on the system.
   clear_ffdc          Clear all FFDC log files on the system.
   clear_tranlog       Clearing transaction logs for application server.
   monitor             Monitor log files on the system.
   Global commands:
   back                Return to the previous command mode.
   exit                Log off from the appliance.
   help                Display information for using the specified command.
   reboot              Reboot the appliance.
   shutdown            End system operation and turn off the power.
   top                 Return to the top level.
   ```

For more information on clearing transaction logs, see Clearing the transaction logs.

7. At the `isimvasrv:logs` prompt, enter **monitor**.
8. At the `isimvasrv: monitor` prompt, enter **help**.
9. Enter the index number to view a list of logs.

```
Options:
1: SystemErr.log
2: SystemOut.log
3: native_stderr.log
4: native_stdout.log
5: startServer.log
6: stopServer.log
```

10. Enter the index number to view the tailing logs of the cluster manager. For example, specify **1**.

    The following message is displayed:

```
************ Start Display Current Environment ************
Log file started at: [3/13/15 17:42:49:673 EDT]
************* End Display Current Environment *************
```

11. Enter the index number to view the tailing logs of the IBM Security Identity Manager Server. For example, specify **7**.

    The following message is displayed:

```
1: SystemErr.log
2: SystemOut.log
3: SystemOut_15.03.17_02.42.19.log
4: native_stderr.log
5: native_stdout.log
6: startServer.log
7: stopServer.log
8: SystemErr.log
9: SystemOut.log
10: native_stderr.log
11: native_stdout.log
12: startServer.log
13: stopServer.log
14: msg.log
15: trace.log
```

12. Enter the index number to view the trace logs of the IBM Security Identity Manager Server. For example, specify **15**.

    The following message is displayed:

```
<Time Millis="1426836005522"> 2015.03.20 03:20:05.522-04:00</Time>
<Server Format="IP">isim1175.in.ibm.com</Server>
<ProductId>CTGIM</ProductId>
<Component>com.ibm.itim.pim.serviceprovider.db</Component>
<ProductInstance>ISIMVa_APP_MEMBER</ProductInstance>
<LogText><![CDATA[ISPIM DB manager is processing asynchronous requests? true]]></LogText>
<Source FileName="PIMDBManager" Method="run"/>
<Thread>Thread-139</Thread>
</Trace>
```

migration command

The migration command provides options to work with the IBM Security Identity Manager migration utilities in case of migrated system.

Usage

CLI:

```
isimvasvr > isim
isimvasvr: isim > migration
isimvasvr: migration >
```

You use the migration related commands during the post-migration production cutover step. For more information, see Production cutover roadmap .

db_migrate
Run database migration scripts for migration from IBM Tivoli Identity Manager 5.1 to IBM Security Identity Manager 7.x.

ldap_migrate
Run LDAP migration scripts for migration from IBM Tivoli Identity Manager 5.1 to IBM Security Identity Manager 7.x.

Examples

- Run database migration scripts.

  At the *isimvasvr*: migration prompt, enter **db_migrate**.

```
This operation will execute the database migration scripts.
Do you wish to continue?
Enter 'YES' to confirm: YES
Executing the migration scripts for database.
Please wait...
Completed the execution of the migration scripts for database. Check log files for more information.
```

- Run LDAP migration scripts.

  At the *isimvasvr*: migration, enter **ldap_migrate**.

```
This operation will execute the LDAP migration scripts.
Do you wish to continue?
Enter 'YES' to confirm: YES
Executing the migration scripts for LDAP.
Please wait...
Completed the execution of the migration scripts for LDAP. Check log files for more information.
```

Important: Restart the IBM Security Identity Manager Server.

nodes_administration command

The **nodes_administration** command provides options to work with the IBM Security Identity Manager application server nodes in the cluster.

Usage

CLI:

```
isimvasvr > isim
isimvasvr: isim > nodes_administration
isimvasvr: nodes_administration >
```

restart
Restarts the IBM Security Identity Manager server node.

start Starts the IBM Security Identity Manager server node.

stop Stops the IBM Security Identity Manager server node. You can use the -force option to stop the servers forcefully.

sync Synchronizes the IBM Security Identity Manager server node with the cluster manager.

Examples

- At the `isimvasrv: nodes_administration` prompt, enter **restart**.

```
This operation restarts all the servers.
Do you wish to continue?
Enter 'YES' to confirm: YES
Stopping cluster manager server...
Stopping node agent server...
Stopping application member server...
Stopping message member server...
Starting cluster manager server...
Starting node agent server...
Starting application member server...
Starting message member server...
Successfully restarted all the servers.
```

- At the `isimvasrv: nodes_administration` prompt, enter **start**.

```
This operation starts all the servers.
Do you wish to continue?
Enter 'YES' to confirm: YES
Starting cluster manager server...
Starting node agent server...
Starting application member server...
Starting message member server...
Successfully started all the servers.
```

- At the `isimvasrv: nodes_administration` prompt, enter the **stop** command.

```
This operation stops all the servers.
Do you wish to continue?
Enter 'YES' to confirm: YES
Stopping cluster manager server...
Stopping node agent server...
Stopping application member server...
Stopping message member server...
Successfully stopped all the servers.
```

- At the `isimvasrv: nodes_administration` prompt, enter the **sync** command.

```
This operation synchronizes the ISIM server node with the cluster manager. Cluster manager
should be running for this operation to complete successfully.
ISIM server will be stopped before synchronization.
Do you wish to continue?
Enter 'YES' to confirm: YES
Stopping ISIM server for synchronization
Stopping node agent server...
Stopping application member server...
Stopping message member server...
Synchronizing ISIM server with cluster manager...
Successfully synchronized the ISIM server with cluster manager.
Starting ISIM server after synchronization.
Starting node agent server...
Starting application member server...
Starting message member server...
Successfully started the ISIM server after synchronization.
```

single_sign_on_settings command

Manage the single sign-on (SSO) domain name for your IBM® WebSphere® Application Server environment.

You can work with domain names that contain a set of hosts to which the single sign-on applies.

Usage

CLI:

```
isimvasvr > isim
isimvasvr: isim > single_sign_on_settings
isimvasvr: single_sign_on_settings >
```

domain_name
Work with domain names for single sign-on.

domain_name commands

Work with domain names for single sign-on.

Usage

CLI:

```
isimvasvr > isim
isimvasvr: isim > single_sign_on_settings
isimvasvr: single_sign_on_settings > domain_name
isimvasvr: domain_name >
```

delete Removes the single sign-on domain names.

set Sets the single sign-on domain names.

show Show the single sign-on domain names.

Examples

- Delete a domain name.

  At the isimvasrv:domain_name prompt, enter **delete**.

```
This operation deletes the Domain name(s) for Single Sign-on.
Do you wish to proceed?
Enter 'YES' to confirm: YES
Deleting Domain name(s).. Domain name(s) deleted successfully.
Restart IBM Security Identity Manager and Cluster to apply the new settings.
```

- Set a domain name.

  At the isimvasrv: domain_name prompt, enter **set**.

```
Enter Domain name(Separate multiple Domain names by comma(,)) : example.com

Setting Domain name(s)..
Domain name(s) updated successfully.
Restart IBM Security Identity Manager and Cluster to apply the new settings.
```

- Show the single sign-on domain names.

  At the isimvasrv: domain_name prompt, enter **show**.

```
Domain name(s) : example.com
```

Important: You must restart the IBM Security Identity Manager Server to apply changes in settings after you run a **delete** or **set** command.

upgrade command

The **upgrade** command provides options to work with IBM Security Identity Manager firmware updates.

Usage

CLI:

```
isimvasvr > isim
isimvasvr: isim > upgrade
isimvasvr: upgrade >
```

For more information about updating firmware from a USB storage device, see "Upgrading the IBM Security Identity Manager virtual appliance from a USB device" in the *IBM Security Identity Manager Installation Guide*.

delete Deletes firmware updates from the system.

install
Installs the available firmware update to the system.

list Lists firmware updates from a USB device.

transfer
Transfers firmware update from a USB device to the system.

Examples

- Delete a firmware update from the system.

  At the *isimvasvr*: upgrade prompt, enter **delete**.

```
Are you sure you want to delete the firmware update from the system ?
Enter 'YES' to confirm: YES
1: 7.0.1-ISS-SIM-FP0009.pkg
Enter index: 1
The firmware update '7.0.1-ISS-SIM-FP0009.pkg' deleted from the system
```

- Install a firmware update on the system.

  At the *isimvasvr*: upgrade prompt, enter **install**.

```
Warning: This operation will require that the appliance is rebooted.
Are you sure you want to update the firmware to the inactive partition ?
Enter 'YES' to confirm: YES
1: 7.0.1-ISS-SIM-FP0009.pkg
Enter index:
```

utilities commands

The **utilities** command provides options to work with IBM Security Identity Manager utilities.

Usage

CLI:

```
isimvasvr > isim
isimvasvr: isim > utilities
isimvasvr: utilities >
```

data_sync
Synchronizes data for the IBM Security Identity Manager Service Center.

db_purge
Cleans up the audit trails in the database.

incr_data_synch
Starts, stops, and queries the status of the Incremental Data Synchronizer.

ldap_clean
Cleans up the recycle bin entries with the expired age limits.

remote-debugging
Allows administrators to enable the remote debugging feature of the WebSphere Application Server.

sap Adds support for the SAP GRC Access Control.

data_sync command

The **data_sync** command provides options to synchronize the access catalog related data, access catalog items for the IBM Security Identity Manager Service Center.

It synchronizes data, access catalog items between the LDAP server and the IBM Security Identity Manager database. The synchronized data is used to search an existing access catalog and to perform various access related operations in IBM Security Identity Manager Service Center.

Usage

CLI:

```
isimvasvr > isim
isimvasvr: isim > utilities
isimvasvr: utilities > data_sync
isimvasvr: data_sync >
```

Options

Sync options

Upgrade
Specify this option when you have existing access related data and **data_sync** is not executed till now.

Maintenance

Select this option to resynchronize data when the connection to the database is lost during a data transaction.

Examples of loss can occur because of network failure, WebSphere Application Server failure, or an unavailable database.

Data types

ConfigData
Default configuration data such as group profiles, organization, service (ISIM service), provisioning policy, and global settings.

AccessCatalog
Access catalog data such as role access, group access, service, and provisioning policy.

All Data that is described by `ConfigData` , `AccessCatalog` values.

Example

Synchronize all access catalog and configuration data for the IBM Security Identity Manager Service Center.

At the `isimvasrv: utilities` prompt, enter **data_sync**.

```
Sync options:
1: Upgrade
2: Maintenance
Enter index: 1
Data types:
1: ConfigData
2: AccessCatalog
3: ALL
Enter index:
3
Running data synchronization utility. Data synchronization was successful.
```

db_purge command

The **db_purge** utility cleans up the audit trail in the database by removing records that are related to completed workflow processes.

Usage

CLI:

```
isimvasvr > isim
isimvasvr: isim > utilities
isimvasvr: utilities > db_purge
isimvasvr: db_purge >
```

The utility is useful for deleting historical workflow audit data, non-workflow audit events and reconciliation reporting entries from the database. It handles only removal, not archiving of these records. Use the utility sparingly to avoid any unforeseen problems.

execute
Execute DBPurge immediately.

schedule
Set DBPurge schedule.

Related information:

Cleaning up the database with the DBPurge utility
It is a good maintenance practice to keep the IBM Security Identity Manager Server database to a manageable size.

incr_data_synch command

The Incremental Data Synchronizer utility provides fast synchronization of data and access control item between the directory server that IBM Security Identity Manager uses and the IBM Security Identity Manager database.

The Incremental Data Synchronizer synchronizes staged reporting data (entities and attributes) and corresponding access control item information for the data.

It manages incremental changes to the data and does the synchronization task only on the changed data. By propagating only the changes since the last synchronization task was done, the Incremental Data Synchronizer can update the staged reporting data quickly.

Usage

CLI:

```
isimvasvr > isim
isimvasvr: isim > utilities
isimvasvr: utilities > incr_data_synch
isimvasvr: incr_data_synch >
```

Show options for the **incr_data_synch** utility.

At the *isimvasvr*:utilities prompt, enter **incr_data_synch**.

```
1: Start (Note: Ensure that the 'changelog' feature is enabled in the directory server instance)
2: Status
3: Stop

Select option:
```

To enable changelogs for your version of IBM Security Directory Server, look for "Change log management for a directory server instance" in the IBM Security Directory Server product documentation.

Note: To manage incremental data sync, select option number to start, view status, or stop the service at the isimvasrv:incr_data_sync prompt. When sync starts, data synchronizes continuously until the **stop** command is executed manually.

Examples

- Start the incremental data synchronization process.

  At the *isimvasvr*:incr_data_synch prompt, enter **1**.

  ```
  Enter ITIM Administrator ID: itim manager
  Enter ITIM Administrator password:
  Enter Base DN for changelog entries: dc=com
  Enter time interval between successive synchronizations in seconds: 10
  Incremental Data Synchronizer started.
  ```

- Check the status of the Incremental Data Synchronizer.

  isimvasvr:incr_data_synchprompt, enter **2**.

  ```
  Incremental Data Synchronizer running
  ```

- Stop the Incremental Data Synchronizer.

  At the isimvasvr:incr_data_synch prompt, enter **3**.

  ```
  Incremental Data Synchronizer stopped.
  ```

ldap_clean command

Remove only those objects that are older than the age limit setting when the Recycle Bin feature in IBM Security Identity Manager is enabled. For example, if the age limit setting is 62 days (the default value), only objects that are older than 62 days can be deleted.

Usage

CLI:

```
isimvasvr > isim
isimvasvr: isim > utilities
isimvasvr: utilities > ldap_clean
isimvasvr: ldap_clean >
```

IBM Security Identity Manager has a feature where you can enable a recycle bin. It is disabled by default.

To enable the recycle bin, edit the **enrole.recyclebin.enable** property in `enRole.properties`.

If enabled, then when you delete IBM Security Identity Manager objects (such as organization units, persons, or accounts), the objects are not immediately removed from the directory server. Instead, they are moved to a recycle bin container in directory server. This feature is useful in many scenarios. For example, to avoid assigning an old user IDs to a new user, the assignment process might check the recycle bin to determine whether an old user ID exists.

Emptying the recycle bin is a separate process, called "garbage collection", that involves manually running a cleanup script.

The Recycle Bin Age Limit, specifies the number of days an object remains in the system's recycle bin before it becomes available for deletion by manually running the cleanup script. The Recycle Bin Age Limit protects objects in the recycle bin from cleanup scripts for the specified length of time.

To specify the age limit, edit the property **enrole.ldapserver.agelimit** in the `enRole.properties` file.

Example

Remove objects from the IBM Security Identity Manager recycle bin.

At the *isimvasvr*: utilities prompt, enter **ldap_clean**.

```
Running ldap clean utility.
Remove old entries in Recycle Bin of Directory Server
LDAP clean is successful.
```

remote-debugging command

The remote-debugging command provides options to enable the remote debugging feature of WebSphere® Application Server.

Usage

CLI:

```
isimvasvr > isim
isimvasvr: isim > utilities
isimvasvr: utilities > remote-debugging
isimvasvr: remote-debugging >
```

This command is helpful for debugging the IBM Security Identity Managerapplication that is running on WebSphere® Application Server.

The debug port is 7777 (the default value).

Note: This feature is used by IBM Support to troubleshoot IBM Security Identity Manager application issues.

disable
Disable remote debugging.

enable Enable remote debugging.

status Determine status of remote debugging.

Examples

- Display help

  At the *isimvasvr*: remote-debugging prompt, enter **help**.

```
Current mode commands:
disable      Disable remote debugging.
enable       Enable remote debugging.
status       Determine status of remote debugging.

Global commands:
back         Return to the previous command mode.
exit         Log off from the appliance.
help         Display information for using the specified command.
reboot       Reboot the appliance.
shutdown     End system operation and turn off the power.
top          Return to the top level.
```

- Disable remote debugging.

  At the *isimvasvr*: remote-debugging prompt, enter **disable**.

```
Disabling remote debugging .....
Successfully disabled remote debugging.
```

- Enable remote debugging.

  At the *isimvasvr*: remote-debugging prompt, enter **enable**.

```
Enabling remote debugging .....
Successfully enabled remote debugging.
```

- Display remote debugging status.

  At the *isimvasvr*: remote-debugging prompt, enter **status**.

```
Remote debugging is enabled.
```

sap command

This command provides support for the notification component of SAP GRC Access Control. The Cron utility allows tasks to be automatically run in the background at regular intervals.

Usage

CLI:

```
isimvasvr > isim
isimvasvr: isim > utilities
isimvasvr: utilities > sap
isimvasvr: sap >
```

delete_cron
Deletes the cron job for the notifier.

install_sap_grc
Installs the notification component for SAP GRC Access Control.

log_config
Configures the log level for SAP GRC Access Control component.

run_notifier
Runs the notification component of SAP GRC Access Control.

Note: You must modify SAPNotify.props file with the correct value for each of the attributes before executing **run_notifier**. You can upload or download SAPNotify.props file from the Custom File Management page. To perform this task, from the Virtual Appliance Local Management Interface Dashboard, browse to **Configure Identity Manager** > **Advanced Configuration : Custom File Management** .

For more information on the SAPNotify.props file, see the "SAP GRC Access Control" document.

set_cron
Sets the cron job for the notifier.

Examples

- Delete the cron job for the notifier.

  At the *isimvasvr*: sap prompt, enter **delete_cron**

  ```
  Enter 'YES' to confirm: YES
  Cron job deleted for SAP_GRC.
  ```

- Install the notification component.

  At the *isimvasvr*: sap prompt, enter **install_sap_grc**.

  ```
  Enter the SAP GRC version: 10
  Copied the SAP GRC 10 JAR to WebSphere profile. Restarting WebSphere..

  WebSphere restarted successfully.
  ```

- Configure log level

  At the *isimvasvr*: sap prompt, enter **log_config**.

```
Enter the SAP GRC version: 10
Enter log level (e.g. ALL, NONE, etc.): ALL
Version set to 10.

Log level set to ALL.
```

- Run the notification component for SAP GRC Access Control.

  At the *isimvasvr*: sap prompt, enter **run_notifier**.

```
Enter the SAP GRC version: 10
Starting Notifier
......
Stopping Notifier
```

- Set the cron job for the notifier.

  At the *isimvasvr*: sap prompt, enter **set_cron**.

```
Enter the SAP GRC version: 10
Enter crontab fields.
1: minute
2: hour
3: day of month
4: month
5: day of week
6: continue

Select option: 1
Enter minute <0 to 59>: 2
Enter crontab fields.
1: minute
2: hour
3: day of month
4: month
5: day of week
6: continue

Select option: 6
Cron job set.
```

system_config command

The **system_config** utility is used to correct system configuration errors that occur during installation.

In the virtual appliance, **system_config**, internally executes **runConfig**.

Examples include errors that occur when you set the user password (System user or EJB user), database repository information, logging information, and so on.

Note: If there are any issues with the IBM Security Identity Manager virtual appliance, IBM Support might suggest that you run the **system_config** utility.

Usage

CLI:

```
isimvasvr > isim
isimvasvr: isim > utilities
isimvasvr: utilities > system_config
isimvasvr: system_config >
```

Example

Correct system configuration for errors that occur during installation:

At the *isimvasvr*: utilities prompt, enter **system_config**.

```
Executing ISIM's system configuration.
Please wait...
Execution completed for ISIM's system configuration.
Restart IBM Security Identity Manager server.
```

Note: Restart IBM Security Identity Manager server to apply any changes in settings.

Chapter 20. Sample configuration response file

Set your configuration parameters for the IBM Security Identity Manager virtual appliance in a response file. After you update the response file with correct values, upload the response file to configure the virtual appliance in the advanced configuration mode.

```
##################################################
#
# You can do initial configuration of IBM Security Identity Manager
# Appliance by using a response file.
# Update the response file with correct values and provide it during the advanced
# mode of Initial configuration wizard.
#
##################################################
#
# Appliance Administrator User Credentials
#
isim.appliance.adminUserPwd=<admin user password>

#
# Identity Data store configuration Properties
#
isim.datastore.dbType=<IBM_DB or ORACLE_DB>
isim.datastore.hostName=<hostname>
isim.datastore.port=50000
isim.datastore.adminUser=db2admin
isim.datastore.adminUserPwd=<admin password>
isim.datastore.user=itimuser
isim.datastore.userPwd=<user password>
isim.datastore.dbName=idmdb
isim.datastore.connection.type=non-ssl
isim.datastore.isOracleServiceName=<true or false>

#
# Directory Server configuration properties
#
isim.ldap.hostName=<hostname>
isim.ldap.port=389
isim.ldap.organization.shortname=org
isim.ldap.organization.name=Organization
isim.ldap.bindDN=cn=root
isim.ldap.bindDNPwd=<password>
isim.ldap.dnLocation=dc=com
isim.ldap.connection.type=non-ssl

#
# Mail Server configuration properties
#
isim.mail.server=localhost
isim.mail.from=admin@in.ibm.com
```

Index

E

F

G

H

I

J

L

M

N

O

P

R

S

T

U

V

W

X

Printed in USA