IBM® Security Identity Manager Web Services API

Table of Contents

1 Introduction

The IBM® Security Identity Manager Web Services front end is an enablement customization to provide a web services based channel to communicate with the IBM Security Identity Manager. The IBM Security Identity Manager product comes with a Java API that can be used from external applications to communicate with the IBM Security Identity Manager. However, it requires a Java Authentication and Authorization Service (JAAS) configuration, and version-specific JAR files from the IBM Security Identity Manager application and the IBM® WebSphere Application Server. The environment must be updated whenever the IBM Security Identity Manager or the IBM WebSphere Application Server patches or upgrades are installed. The IBM Security Identity Manager users often ask for an easier, lightweight and Java independent way to communicate with the IBM Security Identity Manager.
The IBM Security Identity Manager Web Services wrapper provides an easy lightweight communication channel to the IBM Security Identity Manager. The Web Services client does not have the IBM Security Identity Manager or the IBM WebSphere Application Server dependency. The Web Services API exposes user functionality for customers to build custom applications. The Web Services API does not expose administration functionality. Some of the advantages of using the web services wrapper instead of the IBM Security Identity Manager API to communicate with the IBM Security Identity Manager are:

- Footprint is small. Provides a standard web services interface using HTTP/S to communicate with the IBM Security Identity Manager.

- IBM Security Identity Manager or the IBM WebSphere Application Server JAR files in the external application are not needed. An IBM WebSphere Application Server client is also not needed on the client side.

- No JAAS configuration needed to talk to the IBM Security Identity Manager.

- The IBM Security Identity Manager Web Services uses a simple data model. A client can generate the model and client artifacts from the Web Services Description Language (WSDL) without needing any special serializers or deserializers.

- The IBM Security Identity Manager API is abstracted into a functional web services API, so client application impact is minimized during the IBM Security Identity Manager or the IBM WebSphere Application Server upgrades and patches.

- Additionally exposes an API to get forms data from the IBM Security Identity Manager so that a client can get data and rendering information. Form changes in the IBM Security Identity Manager are reflected back in real time.

- Provides a threaded conversation capability by providing a session handle to a client. Provides options for server or client state saving mechanisms, or a pseudo stateless conversation if state cannot be saved between calls.

This document describes the IBM Security Identity Manager Web Services.

2 Architecture

2.1 Architecture overview

The IBM Security Identity Manager Web Services wrapper is a suite of web services that are bundled as a single web application. The IBM Security Identity Manager Web Services web application is packaged as a module of `ITIM.ear` and deployed as part of `ITIM.ear`.

2.2 Architecture components

The IBM Security Identity Manager Web Services suite consists of:

- The IBM Security Identity Manager Web Services web application – bundled with `ITIM.ear` file.
- The IBM Security Identity Manager Web Services client and data model (auto-generated by client from WSDL)

The IBM Security Identity Manager Web Services wrapper is a J2EE web application. The web application contains WSDL (Web Services Description Language) files for each web service in the suite. The WSDL files describe the web services interface and complex data types. You can use WSDL files while developing the Web Services clients.

The IBM Security Identity Manager Web Services implementation is based on the IBM WebSphere Application Server JAX WS 2.1.6 runtime.

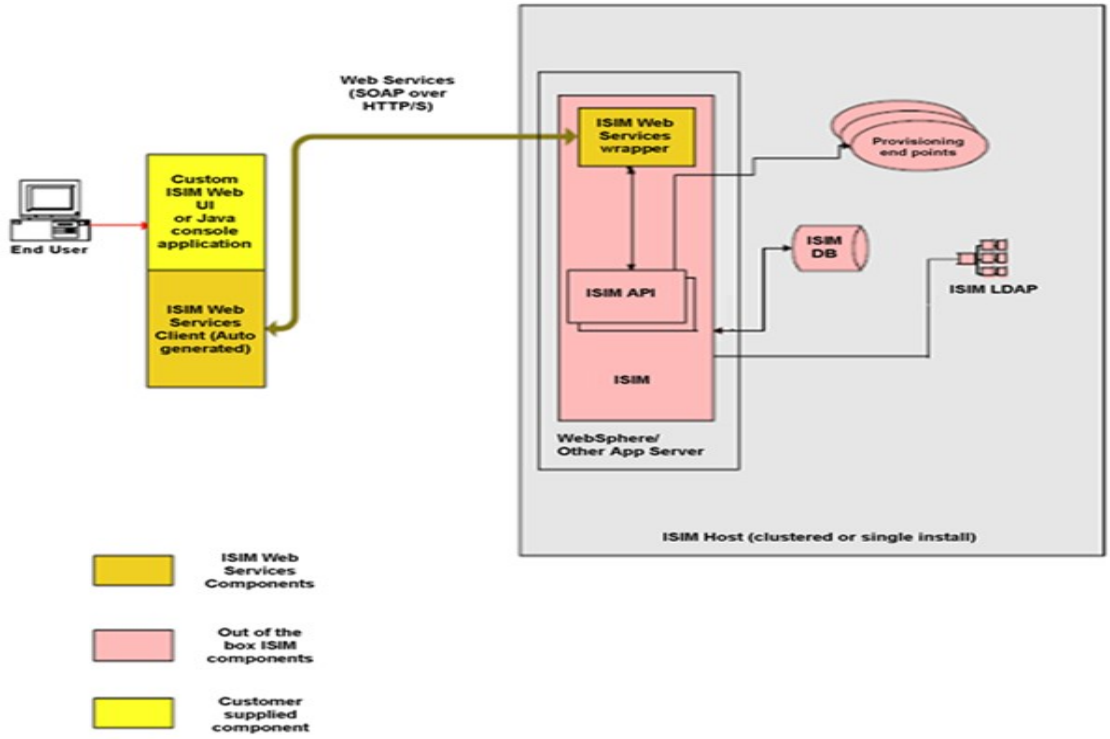


Figure 1: ISIM Web Services Architecture

3 Web Services functionality

The description about the web services suite functionality is as follows.
NOTE: The IBM Security Identity Manager Web Services suite currently only exposes functionality that is required for servicing the end user IBM Security Identity Manager tasks.
The IBM Security Identity Manager Web Services suite consists of multiple web services, broken up by function. The document lists all the services alphabetically except the `WSSessionService` because it is the first service to be called by any application. The session object returned by its login method is used as a parameter in all subsequent services.

The operations list of each web service is as follows.

3.1 WSSessionService

The `WSSessionService` provides authentication, session creation and password challenge authentication methods. This web service login method is called by a client before any of the other web services are invoked. The login method returns a session (handle) object that must be passed to the other web service calls to maintain a threaded conversation. The authentication method also supports Single Sign-On (SSO) if the IBM Security Identity Manager is configured with SSO enabled. The `ISIMWebServices` suite has a handler that intercepts calls into the web services to check valid session information. It also handles session timeouts and cleanup.
This service also provides a method to login through the challenge response authentication process in case of lost or forgotten passwords.

There are two ways in which session management can be done in the ISIM web services.

- Client side session management: This is the default mode. In this mode the session management is not done on the server side. The server does not maintain any session information of the client. The client sends the LTPA token to the server in every request as part of the WSSession object's clientSession attribute. The server then constructs the subject from the clientSession parameter and validates the user. In this scenario there is no support for logout API. The client has to take the responsibility of discarding the session.
- Server side session management: In this mode the sessions are cached on the server side and logout is a valid API.

The property `enrole.webServices.session.mgmt.clientSide in the enrole.properties file enables to switch the modes. A value true means the client mode is enabled.`

`WSSessionService` supports the following methods:

- getChallengeQuestions

Note : Configure Responses before using this API .

- getItimFixpackLevel
- getItimVersion

- getItimVersionInfo
- getProperty
- getWebServicesBuildNumber
- getWebServicesTargetType
- getWebServicesVersion
- isPasswordEditingAllowed (this method returns the system property “Is password editing allowed”
- login
- logout
- lostPasswordLoginDirectEntry (No longer supported)
- lostPasswordLoginResetPassword

`WSSessionService` supports the following methods:

Ø login

This API authenticates the user and returns the session object. This session object is needed to use other web service APIs.

Input:

userID: The user ID of the IBM Security Identity Manager system user.
password: The password of the user.

Output:

session: The `WSSession` object that has the authenticated user information.

Fault:

The fault message is returned with the message key and the message parameters if the user cannot authenticate.

Ø logout

This API logs out the user from the current session.

Input:

session: The session from which the user is to logout.

Output:

None

Fault:

The fault message is returned with the message key and the message parameters if the user can not be logged out.

3.2 WSAccountService

The `WSAccountService` provides methods to perform account related tasks. Other than the basic account operations like create or modify, the service also provides methods to perform related functions like getting default account attributes for a new account as specified by the provisioning policy, or to get the account profile name for a service.
`WSAccountService` supports the following methods:

- adoptAccounts
- adoptSingleAccount
- createAccount
- deprovisionAccount
- getAccountAttributes
- getAccountProfileForService
- getDefaultAccountAttributes
- getDefaultAccountAttributesByPerson
- modifyAccount
- orphanAccounts
- orphanSingleAccount
- restoreAccount
- searchAccounts
- suspendAccount
- updateAccount

3.3 WSGroupService

`WSGroupService` provides methods for group management that were introduced in IBM Security Identity Manager 5.1. These methods allow a user to create groups, remove groups, search for groups, and manage group membership.
The following methods are available:

- addGroupMembers
- createGroup
- getGroupMembers
- getGroupsByAccess

- getGroupsByAccount
- getGroupsByService
- lookupGroup
- removeGroup
- removeGroupMembers

3.4 WSOrganizationalContainerService

`WSOrganizationalContainerService` provides the IBM Security Identity Manager organization tree traversal and retrieval methods.

- createContainer
- getOrganizations
- getOrganizationSubTree
- getOrganizationTree
- searchContainerByAttribute
- searchContainerByName
- searchContainerTreeByAttribute
- modifyContainer
- removeContainer
- lookupContainer

3.5 WSPasswordService

`WSPasswordService` provides password management functionality and supports the following methods:

- changePassword
- generatePassword
- generatePasswordByService
- generatePasswordForService
- getPasswordRules
- getSelfPasswordChangeRules

- isPasswordValid
- joinRules
- selfChangePassword

3.6 WSPersonService

`WSPersonService` provides person object related methods. Apart from simple person operations like create, modify, suspend, restore, and delete, the service also has methods to get a person authorized services (services that a person is entitled to) in the IBM Security Identity Manager or accounts, perform person searches, and get the Principal person object.
`WSPersonService` supports the following methods:

- addRole
- addRolesToPerson
- createPerson
- deletePerson
- getAccountsByOwner
- getAuthorizedPersonProfilesForCreate
- getAuthorizedServices
- getFilteredAccountsByOwner
- getFilteredAuthorizedServices
- getPersonRoles
- getPrincipalPerson
- isCreatePersonAllowed
- isMemberOfRole
- lookupPerson
- modifyPerson
- removeRole
- removeRolesFromPerson
- restorePerson

Note : if password synchronization is enabled , then synchronized password for person will be used , unless synchronized is not set , in this case new password will be set as synchronized password , and then will be used .

- searchPersonsFromRoot
- searchPersonsWithItimAccount
- selfRegisterPerson
- suspendPerson
- suspendPersonAdvanced
- synchGeneratedPassword
- synchPasswords
- transferPerson

3.7 WSProvisioningPolicyService

The `WSProvisioningPolicyService` web service retrieves, creates, modifies, and deletes provisioning policies. See the following sections for its operations.

- createPolicy
- deletePolicy
- getPolicies
- modifyPolicy

3.8 WSRoleService

The `WSRoleService` web service provides the capability to create a static role, modify a static role, get member roles, update the role hierarchy by adding and removing role members, lookup and search roles in the IBM Security Identity Manager.

`WSRoleService` supports the following methods:

- lookupRole
- lookupSystemRole
- searchRoles
- searchForRolesInContainer
- getMemberRoles
- createStaticRole
- modifyStaticRole
- updateRoleHierarchy

3.9 WSServiceService

The `WSServiceService` web service provides functionality related to the IBM Security Identity Manager Services. The service has methods to get supporting data (for example, what is called group data for UNIX, Linux or Windows services), and to check if a password is required when provisioning on a service. It also has a method to get services configured on the IBM Security Identity Manager.
`WSServiceService` supports the following methods:

- getServices
- getSupportingData
- getServiceForAccount

- isPasswordRequired
- lookupService
- searchServices
- testCommunications
- getSupportingDataEntries
- getAccountsForService
- createService
- modifyService

3.10 WSSystemUserService

`WSSystemUserService` provides functionality related to system users. The service also exposes delegation management functionality. `WSSystemUserService` supports the following methods:

- addDelegate
- getChallengeResponseConfiguration
- getDelegates
- getExistingChallengeResponseInfo
- getSystemRoleNames
- getSystemUser
- getSystemUsersForPerson
- modifyDelegate
- removeDelegate
- searchSystemUsers
- setChallengeResponseInfo

3.11 WSSearchDataService

`WSSearchDataService` provides functionality to search various IBM Security Identity Manager directory objects. The search method does not enforce the IBM Security Identity Manager ACIs, however a valid IBM Security Identity Manager session is required to call these methods.

The service supports a generic search method which takes an array of parameters like search base, search type, search filter, and others. It also has methods to

specifically support search requests that are executed while rendering search controls, search matches and search filters on forms.

The service exposes following methods:

- findSearchControlObjects
- findSearchFilterObjects
- getAttributeNames
- getCommonPersonSearchAttributeNames
- searchData
- searchForDelegates
- searchPersonsFromRoot
- searchPersonWithITIMAccount

3.12 WSRequestService

`WSRequestService` provides the BM Security Identity Manager request related functionality. The following methods are supported by `WSRequestService`:

- abortRequest
- getActivities
- getChildProcesses
- getcompletedRequests
- getCompletedRequestsPage
- getPendingRequests
- getProcess
- getRequest
- searchCompletedRequests

3.13 WSToDoService

The `WSToDoService` web service lets a user access pending assignment items, approve or reject assignment items, submit RFI items, and others. The following methods are currently supported by `WSToDoService`:

- approveOrReject
- approveOrRejectGroups
- getAssignmentGroups
- getAssignments
- getItemsInAssignmnetGroup
- getRFI
- getEntityDetail
- submitRFI

3.14 WSAccessService

`WSAccessService` provides functionality to create a user access, get existing user access of a person, remove user access and search access entitlements available to a person. The following methods are currently supported by `WSAccessService`:

- createAccess
- getAccesses
- removeAccess
- searchAvailableAccessEntitlements

3.15 WSUnauthService

The `WSUnauthService` API provides an interface for all the web service APIs that do not require the IBM Security Security Identity Manager authentication. The existing methods belong to the `WSPersonService`, `WSPasswordService`, and `WSSessionService` APIs. The following methods are currently supported by the `WSUnauthService` API:

- getSelfPasswordChangeRules
- joinRules
- selfRegister
- getChallengeQuestions
  Note : Configure Responses before using this API .

- getItimVersion
- getItimVersionInfo
- getItimFixpackLevel
- getWebServicesBuildNumber
- getWebServicesTargetType
- getWebServicesVersion
- lostPasswordLoginResetPassword

3.16 WSExtensionService

`WSExtensionService` provides a framework to extend the existing web services used by customers. The extended services can:

- Create a new operation to expose a new IBM Security Identity Manager API.
- Implement a custom business logic by calling a customer-provided class and method.
- Pass the customer-provided data objects to the custom operation, and pass back the customer-provided data objects returned from the custom operation to the client.
- Use data model for XML translation capabilities to offer a generic transport for customer data objects. The packaging includes XML to object and vice versa utilities for Java 2 Platform, Enterprise Edition (J2EE) or Java. Other environments must use their environment specific XML parsers.

NOTE: The custom business logic is executed on the server; however, all the calls made to the web services go through the service end point interfaces as if the calls were made from a remote client. Do not assume that the calls can be resolved by passing the web services infrastructure.

Overview of WSExtensionService implementation

The `WSExtensionService` has an `extendUsingXML` method that accepts a customer-provided business class name, method name and input parameters, and returns the output. The customer must create the business class and associated data model objects, package them into a .jar file, place them in the class path, and register the business class name by using the `$ITIM_HOME\data\wsExtensions.properties` file. The `WSExtensionService` resolves the business class at run time, calls the specified method and returns the results.

The sections list the new service and the customer-provided artifacts that are required for implementation. The following example provides detailed instructions to create a sample extension.

1. The operation `WSExtensionService.extendWithXML` takes in the following parameters:
   (`WSSession` session, String `extensionClassName`, String `methodName`, String `paramsXML`)

2. The method returns an XML string that represents the returned data object. The `WSExtensionService.extendWithXML` method uses the `$ITIM_HOME\data\wsExtensions.properties` file to search a key that has the value as the business class name as specified by using the `extensionClassName` input parameter. If the key exists, the `extendWithXML` method instantiates the specified business class object, and calls the method. The method that is called must use the following signature:

```
public String customMethodName(WSSessionExt session, String paramsXML)
```

For example,

```
public String GetServiceGroups(WSSessionExt session, String paramsXML)
```

Business class and method (provided by customer)

Place the business class in the class path of the server application and modify the `$ITIM_HOME\data\wsExtensions.properties` file to add a new key with value that matches the name of the business class. Package the business class and associated model objects into a JAR file and add the JAR file as a shared library to the IBM Security Identity Manager application by using the IBM WebSphere Application Server (WAS) Administration Console. If you get "`Class not found`" exceptions, the IBM WebSphere Application Server class loader might not be locating your class. Restart the application after adding the JAR file of your business class to the class path of the IBM Security Identity Manager application.

Example of implementing a new `GetServiceGroups` web service operation:

This example implements a new `GetServiceGroups` web service operation which performs multiple operations and returns a data object that contains the result.

Step 1: Define the business class and method to implement the custom logic.

Define a regular Java class and add the method that implements the custom logic. This example defines a `SampleWSExtension` class that contains a `GetServiceGroups` method. The method must not be static and use following signature:

```
public String methodName(WSSessionExt session, String paramsXML);
```

The method uses the `XMLBeanReader` and `XMLBeanWriter` utility classes to translate between XML and data objects. These utility classes are available in the `itim_ws_model.jar` file. `GetServiceGroups` method uses the Subject and PlatformContext information from the `WSSessionExt` parameter and makes multiple API calls.

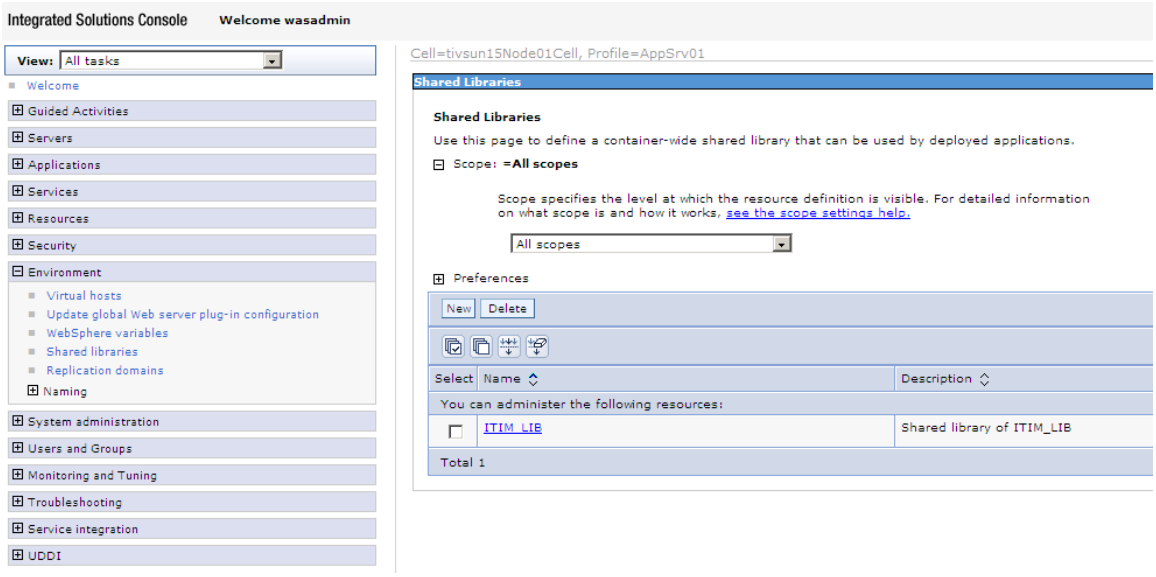
```
public String GetServiceGroups(WSSessionExt session,
        String paramsXML) {
    String retVal = "";
    try {
        Subject subject = session.getSubject();
        PlatformContext platform = session.getPlatform();

        WSPerson person = (WSPerson) XMLBeanReader.readXMLBean(paramsXML,
                WSPerson.class);
        String ownerDN = person.getItimDN();
        String filter = "(&(objectclass=erServiceItem)(owner=" + ownerDN
                + "))";
        ContainerManager containerManager = new ContainerManager(platform,
                subject);
        SearchMO searchMO = new SearchMO(platform, subject);
        searchMO.setContext(containerManager.getRoot());
        searchMO.setCategory(ObjectProfileCategory.SERVICE);
        searchMO.setFilter(filter);

        Collection services = searchMO.execute().getResults();
        GroupManager grpMgr = new GroupManager(platform, subject);
        Locale locale = Locale.getDefault();

        List<String> result = new ArrayList();
        for (Iterator iterator = services.iterator(); iterator.hasNext();) {
            Service service = (Service) iterator.next();
            String servName = service.getName();
            String serviceDN = service.getDistinguishedName().getAsString();
            SearchResultsMO searchResultMO = new SearchResultsMO(platform,
                    subject);
            grpMgr.getGroups(serviceMO, null, "*", searchResultMO, locale);

            Collection groups = null;
            groups = searchResultMO.getResults();

            for (Iterator grpItr = groups.iterator(); grpItr.hasNext();) {
                Group grp = (Group) grpItr.next();
                String grpName = grp.getName();
                result.add(servName + "|" + grpName);
                System.out.println(servName + "|" + grpName);
            }
        }
        retVal = XMLBeanWriter.writeXMLBean(result);
    } catch (Exception e) {
        System.out.println("Exception :");
        e.printStackTrace();
    }
    return retVal;
}
```

The `GetServiceGroups` custom method uses the `XMLBeanReader` utility class to interpret it as a `WSPerson` instance. The method then performs its business logic and returns a List<String> object after converting it to an XML string. The `XMLBeanWrite` utility class performs the XML conversion.

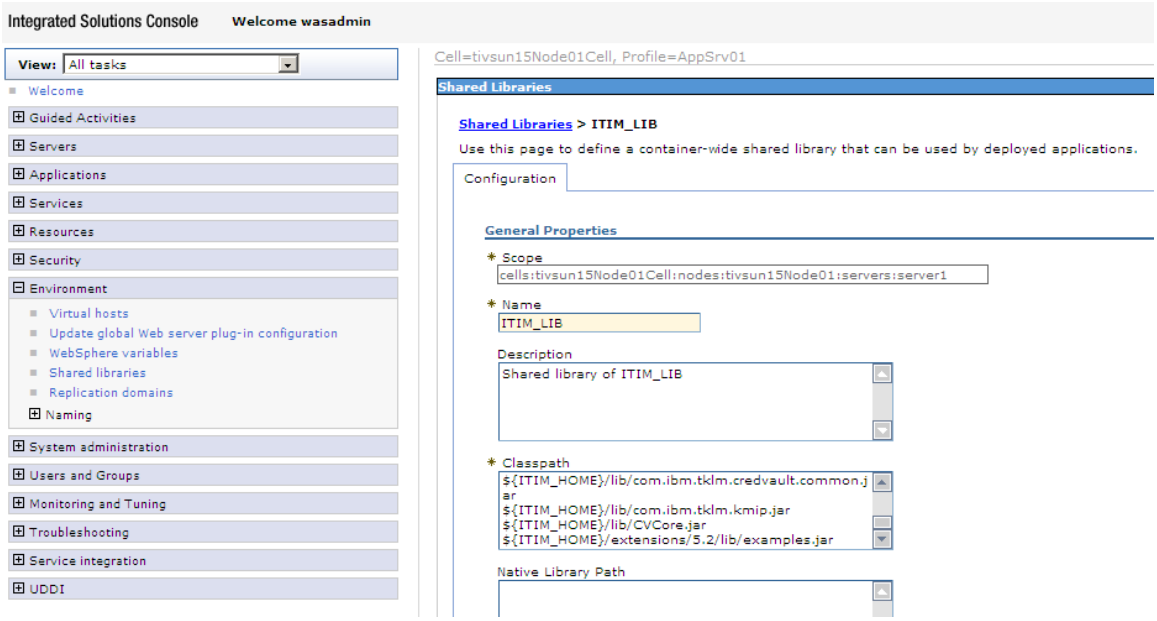
Step 2: Deploy the custom JAR file.

1. Open the `$ITIM_HOME\data\wsExtensions.properties` file by using a text editor.
2. Add a property with a value that matches the business class name and save the file.
   For example, `extension.class1=com.ibm.custom.classname`
3. Add the custom JAR file to the IBM WebSphere Application Server shared library.
4. Create a new shared library at the IBM WebSphere Application Server cell scope level and associate the library with the IBM Security Identity Manager application, or reuse the `ITIM_LIB` shared library that is created during installation of the IBM Security Identity Manager. The following example shows detailed instructions to add custom JAR to the `ITIM_LIB` shared library in IBM WebSphere Application Server 7.0.

1. In the left navigation pane, expand **Environment > Shared libraries** to open the Shared Libraries page, as shown in the following figure:



2. Click `ITIM_LIB` to open the `ITIM_LIB` page.
3. Scroll till the end of the Classpath list and add your custom JAR file. The following example shows the `examples.jar` file that is added to the list. The entry must contain the full path to the JAR file. You might find the `ITIM_HOME` variable to refer to a path in the `ITIM_HOME` directory. (It is not required to place the JAR file in `ITIM_HOME` directory. You can place the JAR file anywhere on the file system visible to the IBM WebSphere Application Server).



4. Click **OK**, and then click **Save** to save the changes to the IBM WebSphere Application Server configuration.
5. Restart the IBM Security Identity Manager application by using the IBM WebSphere Application Server Administration Console.

Step 3: Call the custom web service operation.

The `WSExtensionServiceClient.java` source file in `$ITIM_HOME/extensions/{RELEASE_VERSION}/examples/ws/src` directory demonstrates the sample client code to call the `GetServiceGroups` custom operation.

The `WSExtensionService` supports only the `extendWithXML` method.

3.17 WSSharedAccessService

`WSSharedAccessService` provides many functions for the shared access module that is introduced in IBM Security Identity Manager 6.0. Web service clients must call the login method before calling any other web services. The login method returns a session object that must be passed to the other web service calls in order to maintain a threaded conversation. `WSSharedAccessService` supports the following methods:

- **login**

This API authenticates the user and returns the session object. This session object is needed to use other web service APIs.

Input:

userID: The user ID of the IBM Security Identity Manager system user.
password: The password of the user.

Output:

session: The `WSSession` object that has the authenticated user information.

Fault:

The fault message is returned with the message key and the message

parameters if the user cannot authenticate.

Ø logout

This API logs out the user from the current session.

Input:

session: The session from which the user is to logout.

Output:

None

Fault:

The fault message is returned with the message key and the message parameters if the user can not be logged out.

Ø getAuthorizedSharedAccess

This API gets the authorized credentials and credential pools under the service specified by the unique resource identifier (`serviceURI`) of the service. If the service cannot be found by the service URI the fault message is returned. Both exclusive and non-exclusive credentials are returned and meet these criteria:

Each credential is associated with the corresponding active account.
Exclusive credential are not checked out.

Input:

session: The session for the logged on user.
serviceURI: The unique resource identifier for the service. Internally, this is mapped to the `erserviceuri` attribute of a service.

Output:

One or more `WSSharedAccess` objects is returned. The `WSSharedAccess` complex type has distinguished name, name and description of the credential or credential pool, and describes whether it is a credential or credential pool.

Fault:

The fault message is returned with the message key and the message parameters when the user is not authorized to get the credential or if the application exception is thrown internally by the IBM Security Identity Manager server.

Ø getCredential

This API gets the credential of specified credential component object. The credential component can be a credential or credential pool. If the credential needs to be checked out before getting the credential, this method automatically checks out the credential. If the pool is specified as a credential component, one of the credential from the pool is checked out. When the credential is checked out, the credential is checked out synchronously without the workflow even if the workflow is defined for credential checkout operation.

Input:

session: The session for the logged on user.
credCompDN: The distinguished name of the credential or credential pool.
justification: The justification for checking out the credential.

duration: The number of hours until the checked out credential expires.

Output:

The `WSCredential` object is returned with the following attributes:

userID: The user ID of the credential.

password: The password of the user. A null value is returned if the password is not yet registered for the non-exclusive credential.

leaseInfo: The `WSLeaseinfo` object that has the expiration date and distinguished name of lease object.

isPasswordViewable: The boolean flag that tells whether the password can be shown to the user. This attribute is one of the credential configuration settings.

Fault:

The fault message is returned with the message key and the message parameters when the user is not authorized to get the credential or if the application exception is thrown internally by the IBM Security Identity Manager server.

Ø getCredentialAttributes

This API gets the account attribute values for the credential component object. The credential component can be a credential or credential pool. If the credential needs to be checked out before getting the credential, this method automatically checks out the credential. If the pool is specified as a credential component, one of the credential from the pool is checked out. When the credential is checked out, the credential is checked out synchronously without the workflow even if the workflow is defined for credential checkout operation.

Input:

session: The session for the logged on user.

credCompDN: The Distinguished Name of the credential or credential pool.

attributeNames: The list of account attribute names. For example, "`eruid`" and "`erpassword`" need to be used for the user id and the password.

justification: The justification for checking out the credential.

duration: The number of hours until the checked out credential expires.

Output:

leaseInfo: The `WSLeaseinfo` object that has the expiration and `leaseDN` information. If the credential is checked out, then the lease information is returned.

attributes: The list of the `WSAttribute` objects. The `WSAttribute` has the attribute name and the value pair. If the attribute cannot be found on the account, then the value would not be populated in this list.

Fault:

The fault message is returned with the message key and the message parameters when the user is not authorized to get the credential or if the application exception is thrown internally by the IBM Security Identity Manager server.

Ø checkIn

This API checks in the credential.

Input:

session: The session for the logged on user.
leaseDN: The Distinguished Name of credential lease.

Output:
requestID: The request ID for checking in the credential.

Fault:
The fault message is returned with the message key and the message parameters when the user is not authorized to check in the credential or if the application exception is thrown internally by the IBM Security Identity Manager server.

Ø checkInAllIDs

This API checks in all the checked out credentials that the logged on user has checked out.

Input:
session: The session for the logged on user.

Output:
requestID: The list of request ID for checking in all the credential.

Fault:
The fault message is returned with the message key and the message parameters when the user is not authorized to check in the credential or if the application exception is thrown internally by the IBM Security Identity Manager server.

Ø checkoutWithoutWorkflow

This API checks out the credential of the specified credential component object without using the workflow. The credential component can be a credential or credential pool. If the pool is specified as a credential component, one of the credentials from the pool is checked out. When the credential is checked out, the credential is checked out synchronously without the workflow, even if the workflow is defined for credential checkout operation.

NOTE: The `checkoutWithoutWorkflow` method is provided for automated agents and applications that can not wait for completion of workflow processes. Use this method only when the client cannot use the `checkout()` method. Custom process activities are not executed by the `checkoutWithoutWorkflow` method. If custom process activities are required, you must use the `checkout()` method.

Input:
session: The session for the logged on user.
credCompDN: The Distinguished Name of the credential or credential pool.
justification: The justification for checking out the credential.
duration: The number of hours until the checked out credential expires.

Output:
The `WSCredential` object is returned with the following attributes:
userID: The user ID of the credential.
password: The password of the user. A null value is returned if the password is not yet registered for the non-exclusive credential.
leaseInfo: The `WSLeaseinfo` object that has the expiration date and

distinguished name of lease object.

isPasswordViewable: The boolean flag that tells whether the password can be shown to the user. This attribute is one of the credential configuration settings.

Fault:

The fault message is returned with the message key and the message parameters when the user is not authorized to check out the credential or if the application exception is thrown internally by the IBM Security Identity Manager server.

Ø checkout

This API checks out the credential of the specified credential component object. The credential component can be a credential or credential pool. If the pool is specified as a credential component, one of the credentials from the pool is checked out. If the workflow is defined for the checkout operation the request is submitted to the workflow and the request id is returned. If the workflow is not defined the credential is checked out and the credential information is returned.

When a global life cycle operation is defined to invoke the `checkout()` workflow extension, the checkout of shared accounts is done asynchronously. For this scenario, you must also configure the operation name in the Shared Access Module global setting.

NOTE: If there is no life cycle operation defined to invoke the `checkout()` workflow extension, check out of shared accounts is done synchronously.

Input:

session: The session for the logged on user.

credCompDN: The Distinguished Name of the credential or credential pool.

justification: The justification for checking out the credential.

duration: The number of hours until the checked out credential expires.

Output:

Either `WSCredential` or the `requestID` is returned. If the workflow is defined for the checkout operation, then the request is submitted to the workflow and the request ID is returned. If the workflow is not defined then the `WSCredential` object that has the following attributes is returned:

userID: The user ID of the credential.

password: The password of the user. A null value is returned if the password is not yet registered for the non-exclusive credential.

leaseInfo: The `WSLeaseinfo` object that has the expiration date and distinguished name of lease object.

isPasswordViewable: The boolean flag that tells whether the password can be shown to the user. This attribute is one of the credential configuration settings.

Fault:

The fault message is returned with the message key and the message parameters when the user is not authorized to check out the credential or if the application exception is thrown internally by the IBM Security Identity Manager server.

Ø getAuthorizedSharedAccessByServiceDN

This API gets the authorized credentials and credential pools under the service specified by the distinguished name of the service. If the service cannot be found by the distinguished name of the service the fault message is returned. Both exclusive and non-exclusive credentials are returned and meet these criteria:

Each credential is associated with the corresponding active account.
Exclusive credential are not checked out.

Input:

session: The session for the logged on user.
serviceDN: The distinguished name of a service.

Output:

One or more `WSSharedAccess` objects is returned. The `WSSharedAccess` complex type has distinguished name, name and description of the credential or credential pool, and describes whether it is a credential or credential pool.

Fault:

The fault message is returned with the message key and the message parameters when the user is not authorized to get the credential or if the application exception is thrown internally by the IBM Security Identity Manager server.

Ø getAllAuthorizedSharedAccess

This API gets the authorized credentials and credential pools under the service specified by the unique resource identifier (`serviceURI`) of the service. If the service cannot be found by the service URI the fault message is returned. Both exclusive and non-exclusive credentials are returned and meet these criteria:

Each credential is associated with the corresponding active account.
Exclusive credential including the ones that are already checked out by logged on user or by other users.

Input:

session: The session for the logged on user.
serviceURI: The unique resource identifier for the service. Internally, this is mapped to the `erserviceuri` attribute of a service.

Output:

One or more `WSSharedAccess` objects is returned. The `WSSharedAccess` complex type has distinguished name, name and description of the credential or credential pool, and describes whether it is a credential or credential pool.

Fault:

The fault message is returned with the message key and the message parameters when the user is not authorized to get the credential or if the application exception is thrown internally by the IBM Security Identity Manager server.

Ø getAllAuthorizedSharedAccessByServiceDN

This API gets the authorized credentials and credential pools under the service specified by the distinguished name of the service. If the service cannot be found by the distinguished name of the service the fault message is returned.

Both exclusive and non-exclusive credentials are returned and meet these criteria:

Each credential is associated with the corresponding active account.

Exclusive credential including the ones that are already checked out by logged on user or by other users.

Input:

session: The session for the logged on user.

serviceDN: The distinguished name of a service.

Output:

One or more `WSSharedAccess` objects is returned. The `WSSharedAccess` complex type has distinguished name, name and description of the credential or credential pool, and describes whether it is a credential or credential pool.

Fault:

The fault message is returned with the message key and the message parameters when the user is not authorized to get the credential or if the application exception is thrown internally by the IBM Security Identity Manager server.

4 Example code to use the IBM Security Identity Manager Web Services

The IBM Security Identity Manager Web Services may be accessed by using the supplied WSDL files and auto generating the client, or using the WSDL directly to access the web services.

To get a handle on the Service interface by using JAX WS, adhere to the following code path.

Example1:

```
//Obtaining an handle on the service interface. This is the entry point for
//accessing the WSSessionService methods. The WSSessionService_Service or for that matter all the
//other services have an overloaded constructor where in you can pass in the WSDL URL (i.e. the location
//where the WSDL file is published. It can be a file, ftp or any other URL which obeys the URL
//Specification.


//The default constructor loads the WSDL file from the META-INF/wsdl folder of the client jar file.

WSSessionService_Service service = new WSSessionService_Service();

//Obtaining the port of the service.

WSSessionServicePortProxy port = service.getWSSessionServicePort();

//We can make the WSSessionService operation calls on the port object.

Port.login(username, password);
```

Example 2:

```
//Here we will directly use the WSDL published on the IBM Security Identity Manager server to create the
web service proxy

WSSessionService_Service service = new
WSSessionService_Service(“http://localhost:9080/itim/services/WSSessionService/WEB-
INF/wsdl/WSSessionService.wsdl”, new QName("http://services.ws.itim.ibm.com", "WSSessionService")

WSSessionServicePortProxy port = service.getWSSessionServicePort();
```

ג. Using the pre-compiled Java client.

Once the web service factory class is instantiated, the instance must be reused to get an instance of any of the web services.

```
// Get the Service object
WSSessionService_Service service = new WSSessionService_Service();

// Get an instance of WSSessionService from the web service factory
WSSessionService port = service.getWSSessionServicePort();
```

1.1 Authentication, Challenge Response, and System Information examples

4.1.1 Example: Authenticate to IBM Security Identity Manager and get a session handle

```
String userid = “gverma”;
String password = “secret”;
// Get an instance of WSSessionService from the web service factory
WSSessionService_Service sessionService = new WSSessionService_Service();
// login and get a session
WSSession session = sessionService..getPort().login(userid, credential);
```

4.1.2 Example: Login via challenge response questions for a user (lost password behavior)

```
// Assume that a sessionService is already instantiated as shown in Example 6.1-1
String userid=”gverma”;
Collection criList = new ArrayList(); // List to hold each challenge and response info.
try {
String[] challenges = sessionService.getChallengeQuestions(userid);
for (int i = 0; i < challenges.length; i++) {
WSChallengeResponseInfo cri = new WSChallengeResponseInfo();
cri.setQuestion(challenges[i]);
// At this point, this example assumes that the answer is available in string variable
// “answer” thru user interaction.
cri.setAnswer(answer);
criList.add(cri);
}
WSChallengeResponseInfo[] crInfos =
(WSChallengeResponseInfo[]) criList.toArray(new WSChallengeResponseInfo[criList.size()]);
WSSession session = sessionService.lostPasswordLoginDirectEntry(userid, crInfos);
// Depending on what challenge response behavior is needed, the client can opt to reset the ISIM // service
password instead of direct login via Challenge Response. Uncomment the below line (and // comment the
above statement to get this behavior
// String requestId = sessionService.lostPasswordLoginResetPassword(userid, crInfos);
}
catch (WSLoginServiceException e) {
e.printStackTrace();
}
catch (RemoteException e) {
e.printStackTrace();
}
//……other exceptions omitted
```

Get the IBM Security Identity Manager Version and Fixpack level

```
float itimVersion = sessionService.getItimVersion();
int itimFixpackLevel = sessionService.getItimFixpackLevel();
```

Get the IBM Security Identity Manager Web Services version (informational only)

```
float webServiceVersion = sessionService.getWebServicesVersion();
int webServiceBuildNumber = sessionService.getWebServicesBuildNumber();
```

1.2 Person related task examples

All the examples below assume an `ITIMWebServiceFactory` instance named `webServiceFactory`, and a valid session in a `WSSession` object named session.

4.1.3 Example: Get principal person (get the person object of the logged in person)

```
// Assume that a web service factory instance and session have already been established as shown in Example 4.1.1
WSPersonServiceService personService = new WSPersonServiceService();
WSPerson person = personService.getWSPersonService().getPrincipalPerson(session);
// Now that we have the principal's person object, get the person's name
String name = person.getName();
// get the person's attributes
WSAttribute[] wsAttributes = person.getAttributes();
// Convenience code: Change attributes from Array to a Collection
Collection attributes = Arrays.asList(wsAttributes);
// Get a specific attribute and its value
WSAttribute attribute = WSAttrUtils.getWSAttribute(wsAttributes, "sn");
String lastName = WSAttrUtils.getSingleValue(attribute);
// or
String lastName = attribute.getValues()[0];
```

4.1.4 Example: Create person

```
// This example first searches for an OrganizationalUnit called "Finance", then
// creates a custom person of the type "BluePerson" in that OU.
WSOrganizationalContainerService_Service containerService = new
WSOrganizationalContainerService_Service();
WSPersonService_Service personService = new WSPersonService_Service();
// First, search for an OU called Finance to anchor the person. We set the search to look for org units.
String containerProfile = WSObjectCategoryConstants.ORGUNIT; // Constant choices are // ORGUNIT //
LOCATION // ORGANIZATION (although
// you can use other
// methods to search for
// organizations)
// SECURITY_DOMAIN
String containerName = "Finance"; // Container name to search. You can also use wildcard

// character * as a prefix or suffix.
// Search for container named Finance starting at the root . We use the searchContainerByName method.
The other choices are
// searchContainerByAttribute, getOrganizations, getOrganizationTree and getOrganizationSubTree
methods to get organizational
// containers. In the call to searchContainerByName below, we pass a parent container of null which starts
the search at the
// organizational tree root.
WSOrganizationalContainer[] wsContainers = containerService.searchContainerByName(session, null,
containerProfile, containerName);
if (wsContainers != null && wsContainers.length > 0) {
System.out.println("Found " + wsContainers.length + " containers for " + containerName);
// Set the parent container for the person. If the search found more than 1 container, select
// the one you want. We arbitrarily choose the first found container in this example.
WSOrganizationalContainer parentContainer = wsContainers[0];
// Create a person value object.
WSPerson wsPerson = new WSPerson();
Collection attrList = new ArrayList();
wsPerson.setProfileName("BluePerson"); // IMPORTANT: Set the correct profile name. This
// example uses a custom person entity called
// BluePerson.
// Populate the custom blueId attr
WSAttribute wsAttr = new WSAttribute("blueId", new String[] {"Blue-1022"});
attrList.add(wsAttr);
// Populate the mandatory cn and sn attributes
wsAttr = new WSAttribute("cn", new String[] {"Ben Franklin"});
attrList.add(wsAttr);
```

```
wsAttr = new WSAttribute("sn", new String[] {"Franklin"});
attrList.add(wsAttr);
// Add any more attrs to the Collection attrList, and set attributes on person object.
WSAttribute[] wsAttrs = (WSAttribute[])attrList.toArray(new WSAttribute[attrList.size()]);
wsPerson.setAttributes(wsAttrs);
// Submit a person create request
Calendar calendar = Calendar.getInstance();
calendar.setTime(new Date());
WSRequest request = personService.createPerson(session, parentContainer, wsPerson, calendar);
System.out.println("Submitted person create request id = " + request.getRequestId());
} else {
System.out.println("No container found matching " + containerName);
```

4.1.5 Example: Modify person (including roles)

```
// This example assumes that wsAttributes contains the modified attributes of a person.
String personDN; // This should be set to the DN of the person to be modified.
WSAttribute[] wsAttributes; // this should be set to the modified attributes.
WSRequest request = personService.modifyPerson(session, personDN, wsAttributes);
System.out.println(“Request id of modify person request is : “ request.getRequestId());
```

4.1.6 Example: Suspend person example

```
// The personDN is assumed to contain the DN of the person to be suspended. Assumes that
// a previous search or other operation was made to search for the person to be suspended.
String personDN; // This should be set to the DN of the person to be suspended.
WSRequest request = personService.suspendPerson(session, personDN);
```

4.1.7 Example: Search person example

```
// This method is available as a convenience. The WSSearchDataService provides the
// implementation.
String ldapFilter = “(employeeNumber=12345)”;
String[] attrList = null; // Optional, supply an array of attribute names to be returned.
// A null value will return all attributes.
WSPerson[] persons = personService.searchPersonsFromRoot(session, filter, attrList);
// Print out the person name and DN from the search results
for (int i = 0; i < persons.length; i++) {
WSPerson person = persons[i];
System.out.println( “Name: “ + person.getName() + “, dn: “ + person.getItimDN() );
}
```

4.1.8 Example: Search persons with an IBM Security Identity Manager account

```
// This method is available as a convenience. The WSSearchDataService provides the
// implementation.
String ldapFilter = “(employeeNumber=12345)”;
String[] attrList = null; // Optional, supply an array of attribute names to be returned.
// A null value will return all attributes.
List<WSPerson> lstPerson = personService.searchPersonsWithITIMAccount(session, filter, attrList);
```

4.1.9 Example: Get authorized services for a person (services on which a person is authorized to have a new account)

```
String personDN = personService.getPrincipalPerson(session).getItimDN();
Collecton serviceList = personService.getAuthorizedServices(session, personDN);
// Convenience code to print out the names and DNs of each service

for (Iterator iter = serviceList.iterator(); iter.hasNext(); ) {
WSService service = (WSService) serviceList.next();
System.out.println("Service name is " + service.getName() + " with DN: " +
service.getItimDN());
}
```

4.1.10 Example: Get principal person's roles

```
String personDN = personService.getPrincipalPerson(session).getItimDN();
List<WSRole> roles = personService.getPersonRoles(session, personDN);
```

4.1.11 Example: Check if person is member of role

```
String personDN = personService.getPrincipalPerson(session).getItimDN();
String roleDN; // This should be set to the DN of the role to be checked.
boolean isMember = personService.isMemberOfRole(session, personDN, roleDN);
```

4.1.12 Example: Add role

This operation adds a single role to a person and submit a modify request. To add multiple roles to a person, avoid using the `addRole` operation repeatedly. Instead, add the roles (role DNs) to the "`erroles`" attribute on the `WSPerson` object and submit a single `modifyPerson` operation.

```
Lsit<WSRole> roles = roleService.searchRoles(session, "(errolename=FinanceAdmin)");
// This example assumes that only one role is returned from the search.
String roleDN = roles.get(0).getItimDN();

String personDN = personService.getPrincipalPerson(session).getItimDN();
Calendar calendar = Calendar.getInstance();
calendar.setTime(new Date()); // Set date to current time or to the date / time when the request
// should be submitted
WSRequest request = personService.addRole(session, personDN, roleDN, calendar);
```

4.1.13 Example: Remove role

The `removeRole` operation is very similar to the `addRole` operation. It removes the specified role from the person and submits a modify request to the IBM Security Identity Manager. To delete multiple roles from a person, avoid using the `removeRole` operation repeatedly. Instead, delete the roles (role DNs) from the "`erroles`" attribute on the `WSPerson` object and submit a single `modifyPerson` operation.

4.1.14 Example: Synchronize passwords and synchronize generated password

The `synchPasswords` operation submits a request to synchronize passwords for person accounts. The `synchGeneratedPassword` creates a system generated password that satisfies the password policy and uses that to synchronize the passwords.

```
String personDN = personService.getPrincipalPerson(session).getItimDN();
Calendar calendar = Calendar.getInstance();
calendar.setTime(new Date());
String newPassword="newSecret";
boolean notifyByMail = false; // See usage in synchPasswords – only used in ITIM 5.0 onwards.
// It will be ignored in ITIM 4.6.
// Submit a synchPasswords by specifying the new password
WSRequest syncRequest = personService.synchPasswords(session, personDN,
newPassword, calendar, notifyByMail);
// Or have the system generate a new password and synchronize it.
WSRequest syncRequest2 = personService.synchGeneratedPassword(session,
personDN, calendar);
```

4.1.15 Example: Add roles to person

Assigns the person to more than one roles.

```
WSRequest wsRequest = personService.addRolesToPerson(wsSession, personDN, roleDNlist, date);
```

4.1.16 Example: Remove person from roles

Removes person from the roles.

```
WSRequest wsRequest = personService.removeRolesFromPerson(wsSession, personDN, roleDNlist, date);
```

4.1.17 Example: Transfer person

Transfers a person.

```
WSRequest wsRequest = personService.transferPerson(wsSession, personDN, wsContainer, date);
```

1.3 Account related task examples

All the examples below assume an `ITIMWebServiceFactory` instance named `webServiceFactory`, and a valid session in a `WSSession` object named session.

4.1.18 Example: Create account

```
// Get the account web service.
WSAccountServiceService accountService = new WSAccountServiceService();
// Set the service on which the account is to be created. The authorized services for
// a person can be retrieved by using the WSPersonService's getAuthorizedServices() method.
// Select one of the WSService instances to create the account on.
WSService service; // Set this to the service on which the account is to be created.
String serviceDN = service.getItimDN():
WSAttribute[] wsAttributes; // Set this to contain the account attributes
Calendar calendar = Calendar.getInstance();
calendar.setTime(new Date()); // Set date to current time or to the date / time when the request
// should be submitted
WSRequest request = accountService.getAccountService().createAccount(session, serviceDN,
wsAttributes, calendar);
```

4.1.19 Example: Deprovision account

```
// Get accounts, say by by using the WSPersonService's getAccountsByOwner method.
// Select one of the WSAccount objects to be deprovisioned.
WSAccount account; // Set this to the WSAccount instance to be used.
String accountDN = account.getItimDN();
Calendar calendar = Calendar.getInstance();
calendar.setTime(new Date()); // Set date to current time or to the date / time when the request
// should be submitted
WSRequest request = accountService.deprovisionAccount(session, accountDN, calendar);
```

4.1.20 Example: Get account attributes

```
// Get accounts, say by by using the WSPersonService's getAccountsByOwner method.
// Select one of the WSAccount objects to be deprovisioned.
WSAccount account; // Set this to the WSAccount instance to be used.
String accountDN = account.getItimDN();
List<WSAttribute> attributes = accountService.getAccountAttributes(session, accountDN);
```

4.1.21 Example: Get account profile for service

This is a convenience method provided since the IBM Security Identity Manager API does not have a published way of getting this. It is used internally by the `WSAccountService` `createAccount` method and is published as a convenience. It can also be used to get the form template XML for a new account on a service.

```
String serviceDN; // Set this to the DN of service whose account profile name is needed.
String accountProfile = accountService.getAccountProfileForService(session, serviceDN);
```

4.1.22 Example: Get default account attributes (helpful before provisioning a new account)

This method returns back an array of `WSAttribute` objects that contain the default attribute values as per the provisioning policies in place. At a minimum, it returns the `eruid` attribute (user id) from the identity policy.

```
String serviceDN; // Set this to the DN of the service whose account's default values are sought
List<WSAttribute> defaultAttributes = accountService.getDefaultAccountAttributes(session, serviceDN);
```

4.1.23 Example: Modify account

```
// Get accounts, say by by using the WSPersonService's getAccountsByOwner method.
// Select one of the WSAccount objects to be deprovisioned.
WSAccount account; // Set this to the WSAccount instance to be used.
String accountDN = account.getItimDN();
WSAttribute[] attributes; // Set this to the attributes to be modified
Calendar calendar = Calendar.getInstance();
calendar.setTime(new Date()); // Set date to current time or to the date / time when the request
// should be submitted
WSRequest request = accountService.modifyAccount(session, accountDN, attributes);
```

4.1.24 Example: Restore account

```
// Get accounts, say by by using the WSPersonService's getAccountsByOwner method.
// Select one of the WSAccount objects to be deprovisioned.
WSAccount account; // Set this to the WSAccount instance to be used.
String accountDN = account.getItimDN();
Calendar calendar = Calendar.getInstance();
calendar.setTime(new Date()); // Set date to current time or to the date / time when the request
// should be submitted
String newPassword; // Set this to the new password if a password is required for restore,
// else set to null.
WSRequest request = accountService.restoreAccount(session, accountDN, attributes);
```

4.1.25 Example: Suspend account

```
// Get accounts, say by by using the WSPersonService's getAccountsByOwner method.
```

```
// Select one of the WSAccount objects to be deprovisioned.
WSAccount account; // Set this to the WSAccount instance to be used.
String accountDN = account.getItimDN();
Calendar calendar = Calendar.getInstance();
calendar.setTime(new Date()); // Set date to current time or to the date / time when the request
// should be submitted
WSRequest request = accountService.suspendAccount(session, accountDN, attributes);
```

4.2 Role object related tasks

All the examples below assume an `ITIMWebServiceFactory` instance named `webServiceFactory`, and a valid session in a `WSSession` object named session.

4.2.1 Example: Lookup a role by its DN

```
WSRoleServiceService roleService = new WSRoleServiceService();
String roleDN; // Set this to the DN of the role to be looked up.
WSRole role = roleService.getRoleServicePort().lookup(session, roleDN);
```

4.2.2 Example: Lookup a system role (ISIM Group)

```
WSRoleServiceService roleService = new WSRoleServiceService();
String sysroleDN; // Set this to the DN of the system role to be looked up.
WSRole role = roleService.getRoleServicePort().lookupSystemRole(session, sysroleDN);
```

4.2.3 Example: Search roles by filter

```
WSRoleServiceService roleService = new WSRoleServiceService();
String filter = “(errolename=Finance*)”; // Set a valid LDAP filter
List<WSRole> roles = roleService.getRoleServicePort().lookup(session, filter);
```

4.2.4 Example: Create static role

```
WSRoleServiceService roleService = new WSRoleSerivceService();
roleService.createStaticRole(session, wsContainer, wsRole);
```

4.2.5 Example: Get member roles

```
WSRoleServiceService roleService = new WSRoleSerivceService();
List<WSRole> membRoles = roleService. getMemberRoles(session, roleDN);
```

4.2.6 Example: Update role hierarchy

```
WSRoleServiceService roleService = new WSRoleSerivceService();
roleService. updateRoleHierarchy(session, roleDN, rolesAddedDN[],rolesDeletedDN[],date);
```

4.3 Organizational Container object related tasks

4.3.1 Example: Get Organization sub-tree

```
WSOrganizationalContainerService wsOrgContainerService = this.getWSOrganizationalContainerService();
WSOrganizationalContainer parentWSOrgContainer = this.getParentWSOrgContainer(wsSession,
parentOrg, wsOrgContainerService);
WSOrganizationalContainer wsOrgContainerSubTree =
wsOrgContainerService.getOrganizationSubTree(wsSession, parentWSOrgContainer);
if(wsOrgContainerSubTree != null){
        executedSuccessfully = true;
        printWSOrgContainerDetails(wsOrgContainerSubTree);
}
```

4.3.2 Example: Search container by attribute

```
WSSession wsSession = loginIntoITIM(principle, credential);
WSOrganizationalContainerService wsOrgContainerService = this.getWSOrganizationalContainerService();

WSOrganizationalContainer parentWSOrgContainer = this.getParentWSOrgContainer(wsSession,
parentOrg, wsOrgContainerService);
List<WSOrganizationalContainer> lstWSORganizationalContainers =
wsOrgContainerService.searchContainerByAttribute(wsSession, parentWSOrgContainer, attrName,
attrValue);
if(lstWSORganizationalContainers != null && lstWSORganizationalContainers.size() > 0){
        executedSuccessfully = true;
        //Print Details of the Containers which are returned
        for(WSOrganizationalContainer wsOrgContainer : lstWSORganizationalContainers){
                printWSOrgContainerDetails(wsOrgContainer);
        }
}else{
        System.out.println(" There are no organizational containers matching the attribute name " +
attrName + " and attribute value " + attrValue);
}
```

4.3.3 Example: Create container

```
WSSession wsSession = this.loginIntoITIM(principle, credential);
WSOrganizationalContainer newWSContainer =
createWSOrganizationalContainerFromAttributes(mpParams);
WSOrganizationalContainerService service = this.getWSOrganizationalContainerService();
List<WSOrganizationalContainer> lstOrgContainers = service.searchContainerByName(wsSession, null,
ORG_CONTAINER_PROFILE_NAME, parentOrg);
WSOrganizationalContainer parent = null;
if(lstOrgContainers != null && lstOrgContainers.size() > 0){
        parent = lstOrgContainers.get(0);
}else{
        System.out.println(" Not able to locate the parent container with name " + parentOrg);
}
WSOrganizationalContainer wsOrgContainer = service.createContainer(wsSession, parent,
newWSContainer);
```

4.3.4 Example: Remove container

```
WSSession wsSession = loginIntoITIM(principle, credential);
WSOrganizationalContainerService wsOrgContainerService = this.getWSOrganizationalContainerService();
WSOrganizationalContainer wsContainer = null;
String containerName = (String) mpParams.get(PARAM_ORG_CONTAINER);
if (containerName != null) {
        List<WSOrganizationalContainer> lstWSOrgContainers = wsOrgContainerService
                .searchContainerByName(wsSession, null,"OrganizationalUnit", containerName);
        if (lstWSOrgContainers != null
                && !lstWSOrgContainers.isEmpty()) {
                wsContainer = lstWSOrgContainers.get(0);
        } else {
                System.out.println("No container found matching "
```

```
                        + containerName);
                return false;
        }

} else {
        System.out.println("No Filter parameter passed for the container name.");
        return false;
}
String containerDN = wsContainer.getItimDN();
wsOrgContainerService.removeContainer(wsSession, containerDN);
```

4.3.5 Example: Lookup container

```
WSSession wsSession = loginIntoITIM(principle, credential);
WSOrganizationalContainerService wsOrgContainerService = this.getWSOrganizationalContainerService();
WSOrganizationalContainer wsContainer = null;
String containerName = (String) mpParams.get(PARAM_ORG_CONTAINER);
if (containerName != null) {
        List<WSOrganizationalContainer> lstWSOrgContainers = wsOrgContainerService
                .searchContainerByName(wsSession, null,
                        "OrganizationalUnit", containerName);
        if (lstWSOrgContainers != null
                && !lstWSOrgContainers.isEmpty()) {
                wsContainer = lstWSOrgContainers.get(0);
        } else {
                System.out.println("No container found matching "
                        + containerName);
                return false;
        }
} else {
        System.out.println("No Filter parameter passed for the container name.");
        return false;
}
String containerDN = wsContainer.getItimDN();
WSOrganizationalContainer wsOrgContainer = wsOrgContainerService.lookupContainer(wsSession,
containerDN);
```

1.4 Service object related tasks

4.3.6 Example: Get Services

```
WSSession session = loginIntoITIM(principle, credential);
WSServiceService wsService = getServiceService();
List<WSService> listWsService = wsService.getServices(session);
for (Iterator iterator = listWsService.iterator(); iterator
        .hasNext();) {
        WSService service = (WSService) iterator.next();
        Utils.printMsg(WSServiceServiceClient.class.getName(), "execute", this.name(),
                "\n Service Name : " + service.getName() + " \n " +
                "\n Service Profile Name : " + service.getProfileName() + " \n " +
                "\n Service DN : " + service.getItimDN());
}
```

4.3.7 Example: Get Account for service

```
WSSession session = loginIntoITIM(principle, credential);
WSServiceService wsService = getServiceService();
String serviceName = (String)mpParams.get("serviceName") ;
String filter="(erservicename="+serviceName+")";
String serviceDN = wsService.searchServices(session, null, filter).get(0).getItimDN();
List<WSAccount> listWsAccount = wsService.getAccountsForService(session,serviceDN ,
(String)mpParams.get("accountID"));
for (Iterator iterator = listWsAccount.iterator(); iterator
        .hasNext();) {
        WSAccount account = (WSAccount) iterator.next();
        Utils.printMsg(WSServiceServiceClient.class.getName(), "execute", this.name(),
                "\n Account Name : " + account.getName() + " \n " +
                "\n Account Profile Name : " + account.getProfileName() + " \n " +
                "\n Account DN : " + account.getItimDN());
}
```

4.3.8 Example: Service search

```
WSSession session = loginIntoITIM(principle, credential);
WSServiceService wsService = getServiceService();
List<WSService> listWsService = wsService.searchServices(session, null, (String)mpParams.get("filter"));
for (Iterator iterator = listWsService.iterator(); iterator
        .hasNext();) {
        WSService service = (WSService) iterator.next();
        Utils.printMsg(WSServiceServiceClient.class.getName(),
                "execute", this.name(), "\n Service Name : "
                + service.getName() + " \n "
                + "\n Service Profile Name : "
                + service.getProfileName() + " \n "
                + "\n Service DN : " + service.getItimDN());
}
```

4.3.9 Example: Get service for an account

```
WSSession session = loginIntoITIM(principle, credential);
WSServiceService wsService = getServiceService();
WSAccountServiceService wsaccount = new WSAccountServiceService();
WSAccountService proxy = wsaccount.getWSAccountService();
WSSearchArguments searchArgs = new WSSearchArguments();
searchArgs.setFilter("eruid="+accountID);
String accountDN=proxy.searchAccounts(session, searchArgs).get(0).getItimDN();
WSService service = wsService.getServiceForAccount(session, accountDN);
Utils.printMsg(WSServiceServiceClient.class.getName(),
        "execute", this.name(), "\n Service Name : "
        + service.getName() + " \n "
        + "\n Service Profile Name : "
        + service.getProfileName() + " \n "
        + "\n Service DN : " + service.getItimDN());
```

4.3.10 Example: Create service

```
WSSession wsSession = loginIntoITIM(principle, credential);
WSOrganizationalContainerService wsOrgContainerService = getOrganizationalContainerService();
WSOrganizationalContainer wsContainer = null;
String containerName = (String) mpParams.get(PARAM_ORG_CONTAINER);
mpParams.remove(PARAM_ORG_CONTAINER);
if (containerName != null) {
        List<WSOrganizationalContainer> lstWSOrgContainers = wsOrgContainerService
                        .searchContainerByName(wsSession, null,
                                        "OrganizationalUnit", containerName);
        if (lstWSOrgContainers != null
                        && !lstWSOrgContainers.isEmpty()) {
                wsContainer = lstWSOrgContainers.get(0);
        } else {
                System.out.println("No container found matching "
```

```
                                        + containerName);
                return false;
        }
} else {
        System.out.println("No Filter parameter passed for the container name.");
        return false;
}
String containerDN = wsContainer.getItimDN();
// The remaining input parameters represents service attributes. They will be passed
// to the underlying web service as a list of WSAttributes.
List<WSAttribute> serviceAttributes = getWSAttributesList(mpParams, "CREATESERVICE",
                this.name());

WSServiceService wsServiceServiceObj = getServiceService();
String nameOfCreatedService = wsServiceServiceObj.createService(wsSession,
                containerDN, profileName, serviceAttributes);
```

4.3.11 Example: Modify service

```
WSSession wsSession = loginIntoITIM(principle, credential);
WSServiceService wsService = getServiceService();
String filter = "(erservicename=" + serviceName + ")";
String serviceDN = wsService.searchServices(wsSession, null, filter).get(0).getItimDN();
// The remaining input parameters represents service attributes. They will be passed
// to the underlying web service as a list of WSAttributes.
List<WSAttribute> serviceModifiedAttributes = getWSAttributesList(mpParams,
                "MODIFYSERVICE", this.name());

wsService.modifyService(wsSession, serviceDN, serviceModifiedAttributes);
```

4.4 To Do task related

4.4.1 Example: Get Assignments

```
WSSession session = loginIntoITIM(principle, credential);
WSToDoService wsToDoService = getToDoService();
List<WSAssignment> wsAssignmentList = wsToDoService
                .getAssignments(session);
```

4.4.2 Example: Get RFI

```
WSSession session = loginIntoITIM(principle, credential);
WSToDoService wsToDoService = getToDoService();
WSRFIWrapper wrapperWSRFI = wsToDoService.getRFI(session,
                rfiAssignmentId);
```

4.4.3 Example: Get Entity detail

```
WSSession session = loginIntoITIM(principle, credential);
WSToDoService wsToDoService = getToDoService();
long assignmentId = Long.parseLong(assgnID.trim());
WSEntityWrapper entityWrapper = wsToDoService.getEntityDetail(
                session, assignmentId);
```

4.4.4 Example: Submit RFI

```
WSSession session = loginIntoITIM(principle, credential);
WSToDoService wsToDoService = getToDoService();

long rfiAssignmentId = Long.parseLong(assgnID.trim());
WSRFIWrapper wrapperWSRFI = wsToDoService.getRFI(session,
                rfiAssignmentId);

ArrayOfTns1WSAttribute rfiAttr = wrapperWSRFI.getWsAttrValues();
List<WSAttribute> wsRFIAttr = rfiAttr.getItem();

//Assuming that the RFI is for an account entity
```

```
//and that all the attributes for which the input is requested has string syntax
//Providing a constant string value "RFIVal" for input.
for (WSAttribute attr : wsRFIAttr) {
        String attrName = attr.getName();
        if(!attrName.equalsIgnoreCase("erservice") && !attrName.equalsIgnoreCase("target_dn") && !
attrName.equalsIgnoreCase("container_dn")) {
                ArrayOfXsdString attrVal = new ArrayOfXsdString();
                attrVal.getItem().add("RFIVal");
                attr.setValues(attrVal);
        }
}
wsToDoService.submitRFI(session, wrapperWSRFI);
```

4.4.5 Example: Approve or Reject

```
WSSession session = loginIntoITIM(principle, credential);
WSToDoService wsToDoService = getToDoService();
List<WSAssignment> wsAssignmentList = wsToDoService
                .getAssignments(session);

List<Long> activityIds = new ArrayList<Long>();
for (WSAssignment assignment : wsAssignmentList) {
        activityIds.add(assignment.getId());
}
wsToDoService.approveOrReject(session, activityIds,
                approvalStatus, explanation);
```

4.5 Access Service related task

4.5.1 Example: Create an Access

```
WSSession session = loginIntoITIM(principle, credential);
WSAccessService accessService = getAccessService();
List<WSAccessEntitlement> listWSAccessEntitlement =
accessService.searchAvailableAccessEntitlements(wsSession, null,
                                                personDN, null, accessName);
List<WSNewUserAccess>           wsNewUserAccessLst = new ArrayList<WSNewUserAccess> ();
WSNewUserAccess wnua = new WSNewUserAccess();
// set userid, pwd, accountDN and PersonDN on wnua.
wsNewUserAccessLst.add(wnua);
List<WSRequest> listReq = accessService.createAccess(wsSession,
                                        wsAccessEnt, wsNewUserAccessLst, null);
```

4.5.2 Example: Get Accesses

```
WSSession session = loginIntoITIM(principle, credential);
WSAccessService accessService = getAccessService();
List<WSAccessEntitlement> listWSAccessEntitlement =
accessService.searchAvailableAccessEntitlements(wsSession, null,
                                                personDN, null, accessName);
wsAccessEnt = listWSAccessEntitlement.get(0);
List<WSUserAccess> wsUserAccessList = accessService.getAccesses(
                        wsSession, personDN, wsAccessEnt == null? null: wsAccessEnt.getAccessId());
```

4.5.3 Example: Remove a user Access

```
WSSession session = loginIntoITIM(principle, credential);
WSAccessService accessService = getAccessService();
List<WSPerson> lstWSPersons = personService.searchPersonsFromRoot(
                                        wsSession, personSearchFilter, null);
String personDN = lstWSPersons.get(0).getItimDN();

List<WSUserAccess> wsualist = accessService.getAccesses(wsSession,
                                                                personDN, null);
WSRequest wsreq = accessService.removeAccess (wsSession,
                                                        wsualist.get(0), null);
```

4.5.4 Example: search for available access entitlements

```
WSSession session = loginIntoITIM(principle, credential);
WSAccessService accessService = getAccessService();
List<WSOrganizationalContainer> lstWSOrgContainers =
wsOrgContainerService.searchContainerByName(wsSession, null,
                                                    "Organization", container);
wsContainer = lstWSOrgContainers.get(0);
List<WSPerson> lstWSPersons = personService.searchPersonsFromRoot(
                                                    wsSession, personSearchFilter, null);
String personDN = lstWSPersons.get(0).getItimDN();
                                            List<WSAccessEntitlement> wsAccessEntitlementList =
accessService.searchAvailableAccessEntitlements(wsSession, container,
                                                        personDN, accessType, accessInfo);
```

4.6 UnAuth related task

4.6.1 Example: Get challenge questions

```
WSUnauthService wsUnauthService = getUnauthService();
List<String> challengeQuestions = wsUnauthService.
                                                    getChallengeQuestions(principle, wsLocale);
for (String question : challengeQuestions)
        System.out.println (question);
```

4.6.2 Example: Lost password login reset password

```
List<String> answersList = new ArrayList<String>();
answersList.add(answer);
WSUnauthService wsUnauthService = getUnauthService();
List<String> challenges = wsUnauthService.
                                getChallengeQuestions(principle, wsLocale);
if (challenges.size() != answersList.size())
        return false;
List<WSChallengeResponseInfo> criList = new ArrayList<WSChallengeResponseInfo>();
for (int i = 0; i < challenges.size(); i++) {
        WSChallengeResponseInfo cri = new WSChallengeResponseInfo();
        cri.setQuestion(challenges.get(i));
        cri.setAnswer(answersList.get(i));
        criList.add(cri);
}
String requestId = wsUnauthService.lostPasswordLoginResetPassword(
                                                        principle, criList, wsLocale);
```

4.6.3 Example: Self register

```
WSPerson wsPerson = createWSPersonFromAttributes(inputParamsMapWithPersonAttribs);
WSUnauthService wsUnauthService = getUnauthService();
// tenantId is an optional parameter if not specified then the value specified
// in the enRole.properties for property "enrole.defaulttenant.id" is used.
wsUnauthService.selfRegister (wsPerson, tenantId);
```

4.6.4 Example: Get self password change rules

```
WSUnauthService wsUnauthService = getUnauthService();
WSPasswordRulesInfo wsRuleInfo = wsUnauthService
                                            .getSelfPasswordChangeRules(accountDN);
```

1.5 Extension Service related task

4.6.5 Example: Extend with XML

```
WSSession session = loginIntoITIM(principle, credential);

WSPersonServiceService service = new WSPersonServiceService();
WSPersonService port = service.getWSPersonService();
// Assume that a person with full name as John Smith exists and ensure that
```

```
// and that this person is owner of one or more services in the IBM Security
// Identity Manager.
String filter = "(cn=John Smith)";
List<WSPerson> searchResults = port.searchPersonsFromRoot(
                        session, filter, null);
if (searchResults == null) {
            Utils.printMsg(WSExtensionServiceClient.class.getName(),
                                    "execute", this.name(),
                                    "A person 'John Smith' must exist.");
            return false;
}

WSPerson wsPerson = searchResults.get(0);
// Convert the object to XML string
String xmlParam = "";
xmlParam = XMLBeanWriter.writeXMLBean(wsPerson);
WSExtensionService extensionService = getExtensionService();
// Execute the sample logic to get the service group names of
// the services which 'John Smith' owns
String resultStr = extensionService.extendWithXML(session,
                        "sample.extension.SampleWSExtension",
                        "GetServiceGroups", xmlParam);
List<String> result = (List<String>) XMLBeanReader.readXMLBean(
                        resultStr, List.class);
```

5 SharedAccess related examples

The following example demonstrates how to use the client proxy interface to call the shared access Web Services APIs.

```
// Get the client side proxy to call the shared access Web Services APIs.
URL serviceURL = new URL(
"http:/localhost:9080/itim/services/WSSharedAccessService/WEB-INF/wsdl/WSSharedAccessService.wsdl");
WSSessionService_Service service =
new WSSessionService_Service(url,
new QName(""http://services.ws.itim.ibm.com", WSSharedAccessService));
WSSharedAccessService proxy = servcie.getWSSharedAccess();

// Authenticate and get the session
WSSession session = proxy.login("userID", "password");

// Get authorized shared accesses
List<WSSharedAccess> authorizedSharedAccess =
proxy.proxy.getAuthorizedSharedAccess(session, serviceURI);

// Get the credential
WSCredential wsCredential = proxy.getCredential(session, credCompDistinguished Name, justification,
duration);

// Check in the credential
String requestID = proxy.checkIn(session, leaseDN);

// Log out
 proxy.logout(session);
```

6 Single Sign-On (SSO) implementation

The IBM Security Identity Manager (ISIM) Web services provides a fallback mechanism when authenticating the user. The IBM Security Identity Manager `WSSessionHandler` first verifies the Simple Object Access Protocol (SOAP) message to confirm whether the session details passed in the message can assign it a valid subject. If it does not, then `WSSessionHandler` first looks into the SOAP header for the `WS-Security` header. If `WSSessionHandler` is not able to locate the identity token and hence a valid subject, `WSSessionHandler` then looks for the Lightweight Third Party Authentication (LTPA) token in the HTTP cookie `LtpaToken2`.

The IBM Security Identity Manager Web services support Single Sign-On for its web services. The programming approach relies on the Java Authentication and Authorization Service (JAAS) module to authenticate the token and provide with the appropriate subject with the corresponding principals. The user authenticates using `WSLoginModule` and `WSCallbackHandler`. The code snippet for retrieving the `Subject` from the identity token (LTPA) is the same for `WS-Security` header scenarios, which is described as follows:

```
/**
 * Passing the LTPA token we should be ale to get the subject.
 * @param token
 *          The LTPA token which is passed as part of the SOAP payload. The LTPA token should be decoded before invoking this method.
 * @return
 *          The valid Subject identified by the LTPA token.
 * @throws LoginException
 */
public static Subject getSubjectFromToken(byte[] credToken) throws WSInvalidSessionSOAPFaultException {
    Subject subject = null;
    try{
        //WSCallbackHandler handler = new WSCallbackHandler(credToken);
        CallbackHandler handler = new WSCallbackHandlerImpl(credToken);

        LoginContext loginCtxt = new LoginContext("WSLogin", handler);
        loginCtxt.login();
        subject = loginCtxt.getSubject();
    }catch(LoginException le){
        //le.printStackTrace();
        try{
            throw new WSInvalidSessionSOAPFaultException("The LTPA token is invalid.", "Not able to construct a valid subject from the LTPA token.");
        }catch(SOAPException e){
            //Log the error message saying that error constructing the SOAP Fault exception body.
        }

    }


    return subject;

}
```

Following are the supported SSO use cases or scenarios:

6.1 WS-Security headers

The <Security> header block in a SOAP message provides a mechanism to attach security-related information that is targeted at a specific receiver or the SOAP actor. The LTPA token is an identity token. You can use the standard Web services security header to send this token. The BinarySecurityToken element contains the LTPA token, which the IBM Security Identity Manager Web services consume for authenticating the user. The LTPA token is sent as part of every request that the client makes for invoking the Web services API. The handling of the LTPA token at the client side is out of the scope of this document. See the examples about how you can embed the LTPA token in the BinarySecurityToken element.

The IBM Security Identity Manager Web services provide their own actors. The client sends the <wsse:Security> header with the IBM Security Identity Manager actor to enable the IBM Security Identity Manager to process the header and retrieve the LTPA token.

The IBM Security Identity Manager Actor Identifier URL is:
http://services.ws.itim.ibm.com/60/actor

The actor has versioning support.

The BinarySecurityToken element also supports encoded token in the SOAP header. This version of IBM Security Identity Manager Web services provides for Base64 encoding.

Following is the sample SOAP message with the LTPA token for the Web services API call WSPersonService.getPrincipalPerson:

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
      <S:Header xmlns:wsse="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecuritysecext-1.0.xsd">
            <wsse:Security
actor="http://services.ws.itim.ibm.com/60/actor">
                  <wsse:BinarySecurityToken ValueType="wsst:LTPA">
                        gn0DxEOu/Nn4b9gGr5rVKwpazJis9CRauQ0zfwf0wSRlgQkw
vFON13tnWinWF==
                  </wsse:BinarySecurityToken>
            </wsse:Security>
      </S:Header>
      <S:Body>
            <getPrincipalPerson>
                  <session>
                        <clientSession xsi:nil="true"></clientSession>

      <enforceChallengeResponse>false</enforceChallengeResponse>
                        <locale xsi:nil="true"></locale>
                        <sessionID>0</sessionID>
                  </session>
            </getPrincipalPerson>
      </S:Body>
<S:Envelope>
```

6.2 IBM® Security® Access Manager (ISAM) WebSEAL HTTP headers

IBM Security Access Manager (ISAM) is a complete authorization and network security policy management solution. ISAM WebSEAL is the resource manager responsible for managing and protecting Web-based information and resources. WebSEAL acts as a reverse Web proxy by receiving HTTP or HTTPS requests from a Web browser. WebSEAL delivers content from its own Web server or from junctioned Web application servers. The IBM Security Access Manager authorization service evaluates requests by passing through WebSEAL to determine whether the user is authorized to access the requested resource. WebSEAL provides SSO capabilities.

There are two scenarios for using ISAM webSeal.

- ➔ Iv-user and TAI:

  The Trust Authentication Interface (TAI) approach looks for the iv-user HTTP header in the request which is forwarded by ISAM webSeal. The steps for configuring ISAM and TAI are detailed in the infocenter section “Configuring IBM Security Identity Manager for single sign-on with WebSphere Trust Association Interceptor and Tivoli Access Manager WebSEAL”.

- ➔ LTPA token:

  ISAM WebSEAL sends the identity token (that is the LTPA token) in the HTTP headers. ISAM WebSEAL cannot embed the tokens into the SOAP message, which is the HTTP payload.

You can configure the cookie name which has the identity token in the enRole.properties file. The data folder of the IBM Security Identity Manager installation directory contains this file. The property name is enrole.webseal.ltpa.cookie.name. The default value is LtpaToken2 since the IBM WebSphere Application Server till version 7.x provides this cookie name when WebSEAL sends the request to the protected server. Once you extract the token, submit the token to the JAAS module to authenticate and retrieve the Subject. The code snippet at the start helps to understand the LTPA token authentication.

NOTE: You can configure ISAM WebSEAL to send the LTPA tokens in the cookies. You can refer to the configuration details in the *ISAM WebSEAL administration guide*.

6.3 IBM Security Identity Manager Web Services API best practices

The web services API are asynchronous in nature which is similar to the behavior of the Java API. This enables ISIM to handle multiple requests for different operations from multiple users. The requests are lined up in a queue and there is no guaranteed order in which the requests will be executed. ISIM does not support transaction management on the requests. The underlining data sources namely LDAP and RDBMS support transaction management.

For example: A create person request is submitted by a user. At the same time a new request for changing the attribute is triggered. There is no guarantee which request will be picked up first by ISIM. If the latter one is picked it would fail as there is no user existing.

In scenarios where there are dependency operations to be performed the Request object should be used to ensure that the previous operation is successfully completed and then proceed with the next one. This ensures that there is no race condition which may lead to failures. The WSRequestService can be used to determine the status of the process/operation.

6.3.1 Don’t use ‘GregorianCalendar.setTime(new Date())’ for scheduling arguments .

Several WebServices functions accept javax.xml.datatype.XMLGregorianCalendar parameter used to schedule the action for some time in the future. If the desire is to have the operation start immediately, use null.
Example:

```
GregorianCalendar calendar = new GregorianCalendar();

calendar.setTime(new Date());

XMLGregorianCalendar date =
DatatypeFactory.newInstance().newXMLGregorianCalendar(calendar);
```

Impact:

Passing in a GregorianCalendar.setTime(new Date()) value has two downsides. The first is that if the API call is done on a remote system, the current system’s date may not

match up with the server's date due to being out of sync or timezone differences resulting in undesired behavior. The second, and more important, downside is that specifying a date instead of null results in a message for future activity being added to the SCHEDULED_MESSAGE table creating unnecessary work for ISIM and possibly resulting in lock contention for busy systems.

6.3.2 LDAP Attribute filter

Several WebService functions accept a LDAP filter to search the entities in the LDAP repository. Ensure that a valid and optimal LDAP filter is passed to these functions.

Impact:

Filters used with WSPersonService, WSAccountService should be optimal to minimize the amount of data returned decreasing the time required to return the results.

6.3.3 Specify return attributes

When searching for an entity, explicitly specifying the desired attributes will improve performance.

Impact:

By specifying a list of desired attributes, the directory server can minimize the amount of data returned decreasing the time required to return the results. Specifying an attribute list can also reduce the memory overhead of the resulting data set.