IBM® Security Privileged Identity Manager

Web Services API

Last updated: June 2015

Table of Contents

1 Introduction

The IBM® Security Privileged Identity Manager Web Services front end provides a web services based channel to communicate with the IBM Security Privileged Identity Manager.

The Web Services API exposes user functionality for customers to build custom applications. The Web Services API does not expose administration functionality.

This document describes the IBM Security Privileged Identity Manager Web Services.

Note: To manage credential pools and identity providers, see the IBM Security Privileged Identity Manager REST API developer reference at http://www.ibm.com/support/docview.wss?uid=swg21903311.

2 Web Services functionality

The IBM Security Privileged Identity Manager Web Services suite exposes functionality that is required for servicing the IBM Security Privileged Identity Manager end user tasks.

The IBM Security Privileged Identity Manager Web Services suite consists of multiple web services that are broken up by function. The `WSSessionService` is the first service to be called by any application. The session object that is returned by its login method is used as a parameter in all subsequent services.

2.1 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 Privileged 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 ISPIM 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 Privileged 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.

2.2 WSOrganizationalContainerService

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

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

2.3 WSPasswordService

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

- changePassword
- generatePassword
- generatePasswordByService
- generatePasswordForService
- getPasswordRules
- getSelfPasswordChangeRules
- isPasswordValid
- joinRules

- selfChangePassword

2.4 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 Privileged 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

2.5 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 Privileged Identity Manager.

`WSRoleService` supports the following methods:

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

2.6 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

2.7 WSSearchDataService

`WSSearchDataService` provides functionality to search various IBM Security Privileged Identity Manager directory objects. The search method does not enforce the IBM Security Privileged Identity Manager ACIs, however a valid IBM Security Privileged 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

2.8 WSRequestService

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

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

2.9 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

2.10 WSUnauthService

The `WSUnauthService` API provides an interface for all the web service APIs that do not require the IBM Security Privileged 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

2.11 WSSharedAccessService

`WSSharedAccessService` provides many functions for the shared access module. 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:

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 Privileged 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 Privileged 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 Privileged 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 Privileged 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 Privileged 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 Privileged 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 Privileged 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 Privileged 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 Privileged 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 Privileged 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 Privileged Identity Manager server.

3 Example code to use the IBM Security Privileged Identity Manager Web Services

The IBM Security Privileged 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 see or build code samples, download the SDK package that is included with your IBM Security Privileged Identity Manager download.

Version 2.0 Download: http://www.ibm.com/support/docview.wss?uid=swg24038026
Version 2.0.1 Download: http://www.ibm.com/support/docview.wss?uid=swg24040010

Examples of some WSDL URLs:

WSSessionService
isim.wsdl.url.session=http://localhost:9080/itim/services/WSSessionService/WEB-INF/wsdl/WSSessionService.wsdl

WSRequestService
isim.wsdl.url.request=http://localhost:9080/itim/services/WSRequestServiceService/WEB-INF/wsdl/WSRequestService.wsdl

WSPersonService
isim.wsdl.url.person=http://localhost:9080/itim/services/WSPersonServiceService/WEB-INF/wsdl/WSPersonService.wsdl

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 Privileged 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();
```

هـ. 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

Example: Authenticate to IBM Security Privileged 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);
```

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 ISPIM //
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 Privileged Identity Manager Version and Fixpack level

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

Get the IBM Security Privileged 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.

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];
```

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);
```

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());
```

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);
```

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() );
}
```

Example: Get principal person's roles

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

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);
```

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);
```

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 Privileged 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.

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);
```

Example: Add roles to person

Assigns the person to more than one roles.

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

Example: Remove person from roles

Removes person from the roles.

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

Example: Transfer person

Transfers a person.

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

3.1 Role object related tasks

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

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);
```

Example: Lookup a system role (ISPIM 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);
```

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);
```

Example: Create static role

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

Example: Get member roles

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

Example: Update role hierarchy

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

3.2 Organizational Container object related tasks

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);
}
```

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);
}
```

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);
```

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);
```

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);
```

3.3 To Do task related

Example: Get Assignments

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

Example: Get RFI

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

Example: Get Entity detail

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

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);
```

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);
```

3.4 UnAuth related task

Example: Get challenge questions

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

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);
```

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);
```

Example: Get self password change rules

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

4 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);
```

5 Single Sign-On (SSO) implementation

The IBM Security Privileged Identity Manager (ISPIM) Web services provides a fallback mechanism when authenticating the user. The IBM Security Privileged 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 Privileged 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;

}
```

The following section describes the supported SSO use cases or scenarios:

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

The IBM Security Privileged 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 Privileged 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/Nn4b9gGr5rVKwpazJis9CRauQ0zfwf0wSRlgQkwvFON13tnWinWF==
                  </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>
```

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 IBM Security Privileged Identity Manager 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. IBM Security Privileged Identity Manager does not support transaction management on the requests. The underlying data sources such as 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 IBM Security Privileged Identity Manager. 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.

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 IBM Security Privileged Identity Manager and possibly resulting in lock contention for busy systems.

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 should be optimal to minimize the amount of data returned decreasing the time required to return the results.

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.

6 Change history

Dec 2014	Initial.
June 2015	Removed WSServiceService, WSAccountService, WSGroupService. Added WSDL URL examples.

Table 6.1: Change history

7 Notices

This information was developed for products and services offered in the U.S.A.

IBM® may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing

IBM Corporation

North Castle Drive

Armonk, NY 10504-1785 U.S.A.

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing

Legal and Intellectual Property Law

IBM Japan, Ltd.

19-21, Nihonbashi-Hakozakicho, Chuo-ku

Tokyo 103-8510, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law :

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement might not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation

2Z4A/101

11400 Burnet Road

Austin, TX 78758 U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurement may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are

fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. _enter the year or years_. All rights reserved.

If you are viewing this information in softcopy form, the photographs and color illustrations might not be displayed.

Terms and conditions for product documentation

Permissions for the use of these publications are granted subject to the following terms and conditions.

Applicability

These terms and conditions are in addition to any terms of use for the IBM website.

Personal use

You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these publications, or any portion thereof, without the express consent of IBM.

Commercial use

You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

Rights

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in

its discretion, the use of the publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

Trademarks

IBM, the IBM logo, and ibm.com® are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at http://www.ibm.com/legal/copytrade.shtml.

Adobe, Acrobat, PostScript and all Adobe-based trademarks are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, other countries, or both.

IT Infrastructure Library is a registered trademark of the Central Computer and Telecommunications Agency which is now part of the Office of Government Commerce.

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

ITIL is a registered trademark, and a registered community trademark of the Office of Government Commerce, and is registered in the U.S. Patent and Trademark Office.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Java™ and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Cell Broadband Engine is a trademark of Sony Computer Entertainment, Inc. in the United States, other countries, or both and is used under license therefrom.

Linear Tape-Open, LTO, the LTO Logo, Ultrium, and the Ultrium logo are trademarks of HP, IBM Corp. and Quantum in the U.S. and other countries.

Privacy Policy Considerations

IBM Software products, including software as a service solutions, (“Software Offerings”) may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering’s use of cookies is set forth below.

This Software Offering uses other technologies that collect each user's user name, password or other personally identifiable information for purposes of session management, authentication, single sign-on configuration, usage tracking, or functional purposes. These technologies can be disabled, but disabling them will also eliminate the functionality they enable.

This Software Offering does not use cookies to collect personally identifiable information. The only information that is transmitted between the server and the browser through a cookie is the session ID, which has a limited lifetime. A session ID associates the session request with information stored on the server.

If the configurations deployed for this Software Offering provide you as customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, see IBM’s Privacy Policy at http://www.ibm.com/privacy and IBM’s Online Privacy Statement at http://www.ibm.com/privacy/details/us/en sections entitled "Cookies, Web Beacons and Other Technologies" and "Software Products and Software-as-a Service".