# SailPoint IdentityIQ Connector Factory

Version 6.4

# Developer's Guide

# Table of Contents

# Chapter 1: Introduction

This chapter discusses the following topics:

## About this book

This book is intended for developers of SailPoint IdentityIQ Connector. It describes how to design, implement, and test a Connector developed for a specific Managed System.

The Connector SDK is used by:

- Managed System vendors who want to provide their Managed System a Connector to enable it to interface with IdentityIQ.
- Independent software developers who want to develop and market a new Connector to connect to a Managed System for which an out-of-the-box Connector is not available.
- Customers who want to connect IdentityIQ to one of their Managed Systems for which an out-of-the-box Connector is not available.

## Definitions, acronyms, and abbreviations

This section describes the definitions, acronyms, and abbreviations used for the IdentityIQ Provisioning Services Manager. Shortened or alternative names appear in parentheses.

**Connector:** The interface between a SailPoint IdentityIQ and one or more Managed Systems.

**Connector Factory:** A user interface that includes guided tasks for developing the new Connector.

**IdentityIQ Provisioning Services Manager wizard (New Project wizard):** Part of the Connector Factory user interface. A preliminary questionnaire for generating definitions of the new Connector.

**Deployment Connector:** A set of files generated by Connector Factory. These files are used to install your customized Connector on the computer where your Managed System is located and to import the new Managed System type into IdentityIQ.

**IdentityIQ:** The central management component of CONTROL-SA, SailPoint Software's solution to the problem of security administration.

**Framework:** A framework of services (such as configuration parameter retrieval and debugging display) provided by a Connector.

**Guided Development Environment (GDE):** Part of the Connector Factory user interface which enables developing the functionality defined in the New Project wizard and generating a deployment Connector.

**Managed System:** The vendor product that is provisioned by SailPoint IdentityIQ (for example, Sun™ Solaris™ or Oracle™ Applications).

# What is a Connector?

This book describes the services provided by a Connector. Connector services provide the interface between a SailPoint IdentityIQ and one or more Managed Systems (see Figure 1— SailPoint IdentityIQ Components on page 7).

A Managed System is a computer operating system, or any application such as a database, that uses an Account ID and password to determine if access permissions should be granted.

As an alternative to Account ID and password login credentials, a Managed System may use another authentication system such as: challenge/response, or two-factor secure log in.

The Connector manages the lifecycles of Entities through the services provided by the Managed System. The available Entity management operations typically are: add, update, revoke, delete.

Examples of Managed Systems are:

- The native security component of an operating system (for example, Microsoft Windows 2000/2003, Solaris, HP-UX, Novell NetWare).
- A directory Service (Microsoft Active Directory)
- A database (for example, Oracle™).

A separate Connector must be designed and implemented for each type of Managed System (for example, an Oracle database). Each Connector provides the functionality required to interface with only one type of Managed System. SailPoint Technologies has available a large number of standard, out-of-the-box Connectors.

A Connector is implemented as a plug-in to IdentityIQ. IdentityIQ installed on a platform provides a common interface to IdentityIQ for different Managed System-specific Connectors installed on the same platform.

For example, IdentityIQ installed on a Microsoft Windows computer can support different Connectors for the following types of Managed Systems:

- The Windows operating system
- Active Directory
- Oracle database
- Any Windows-based application that exposes provisioning and access management services

Figure 1— SailPoint IdentityIQ Components on page 7 illustrates the relationship between IdentityIQ, Connectors, and Managed Systems. Each Managed System installed on a specific platform (for example, Microsoft Windows 2003 or various flavors of UNIX) can communicate between IdentityIQ and the different Connector(s) only for that specific platform.

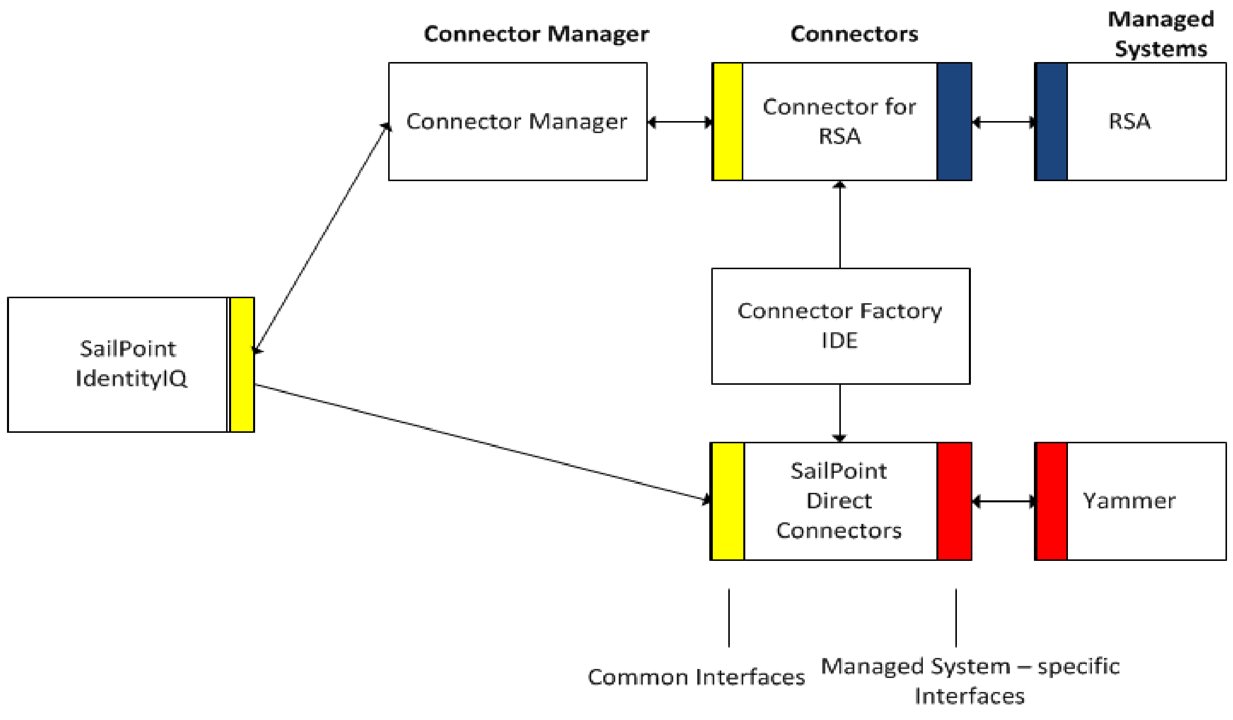

Figure 1—SailPoint IdentityIQ Components

For more information about SailPoint IdentityIQ components, see Chapter 2, "IdentityIQ Connector Factory architecture."

## Related documents

It is highly recommended that you review SailPoint IdentityIQ Administration Guide before designing and implementing a Connector.

**Related documents**

# Chapter 2: IdentityIQ Connector Factory architecture

This chapter discusses the following topics:

## SailPoint IdentityIQ Connector Factory architecture

This section provides an overview of the IdentityIQ Connector Factory architecture.

### Overview

The Connector is constructed of two parts:

- **IdentityIQ Connector Factory Framework:** which interfaces with the IdentityIQ SDK providing a simple SDK in Java for the Connector writer
- **Connector**: this is what the developer implements using the IdentityIQ Connector Factory API for management of the Managed Systems

The SailPoint IdentityIQ Connector Factory is used to create and package the Connector.

### Connector Factory

Connector Factory tool is a software application that provides a friendly, easy-to-use interface with guided processes for developing your Connector.

Connector Factory is the primary development tool for the Connector. It simplifies the process of designing and developing a Connector by breaking down the development process into clearly identifiable steps.

At the end of this process Connector Factory generates a deployment connector for deploying the Connector.

## Key features of Connector Factory

- **Guided Processes:** Connector Factory leads you step-by-step through the process of designing and developing the Connector required to integrate your Managed System into SailPoint IdentityIQ.
- **Ease of Design:** A startup wizard guides you through the process of setting up a new project.
- **Choice of Development Language:** Connector Factory allows you to develop the functionality required by the Connector in Java programming language.
- **Ease of Implementation:** The Connector Factory's Guided Development Environment guides you through the process of creating appropriate Keywords and parameters and writing the function code required by your Connector.

  Skeletons for the code of each required function are provided. You can edit function code using the built-in editor, or you can use any other editor you prefer.
- **Choice of Development Editor:** You can develop function connectors using the built-in editor, or you can choose to employ any other text editor or development environment that you prefer.
- **Integrated Testing Environment:** You can test your new functions from within Connector Factory without having to compile and deploy new Connectors.
- **Ease of generation:** Connector Factory generated connectors are totally ready for deployment.

For more information about the SailPoint IdentityIQ Connector Factory, see the *SailPoint IdentityIQ Connector Factory User Guide*.

# Chapter 3: Designing a new Connector

This chapter presents the following topics:

## Overview

This chapter presents a step-by-step process for designing and creating a Connector. These steps are:

- Understand SailPoint IdentityIQ
- Understand your Managed System and how SailPoint IdentityIQ will interact with it
- Prepare and design your Connector
  - Choose a programming language
  - Decide which types of entities the Connector will manage
  - Decide which types of functionality the Connector will provide for each type of managed entity
  - Choose a Connector type, based on the functionality that you want the Connector to provide
  - Determine which, if any, user-defined Keywords and parameters will be necessary to implement the required functionality

After you have done this, you will be ready to use Connector Factory to develop the connector and bring it into production use. These tasks are described in more detail in the *SailPoint IdentityIQ Connector Factory User Guide*, and include:

- Review the *SailPoint IdentityIQ Conntor Factory User Guide* to understand how to use the Connector Factory tool
- Install Connector Factory
- Use Connector Factory to
  - Define your project details using the Connector Factory New Project wizard
  - Program the required functions, using the Connector Factory Guided Development Environment
  - Test the Connector, using the Testing step in the Connector Factory Guided Development Environment
  - Generate the Connector
- Import the new Managed System type into IdentityIQ
- Test and debug the Connector

## Preparation and design

This section describes the preparation and the design concepts of a Connector.

## Preparation

The preparation stage consists of:

### Understand SailPoint IdentityIQ

Before you begin to design and develop a Connector it is extremely important that you be familiar with SailPoint IdentityIQ architecture, elements, and entities.

Review the background material on SailPoint IdentityIQ in "SailPoint IdentityIQ Connector Factory architecture" on page 9.

### Understand the Managed System

The Managed System is a security product or connector, which may or may not reside on IdentityIQ. The Managed System may be the native security of an operating system (for example, Solaris™, HP-UX, or Novell NetWare) or any other product that implements security (for example, SAP, Oracle™, or Microsoft SQL Server). While the Managed System can reside anywhere in the network, it is managed via the IdentityIQ platform.

Each Managed System has its own Managed System database, which contains security administration data for that Managed System. To enable external applications (such as the Connector) to access the Managed System and perform functions, the Managed System must provide an API.

#### *Supported Programming language*

Connector Factory supports Java programming language.

**Note:** **IdentityIQ Connector Factory does not provide code validation. It is the responsibility of the developer to ensure that the function code is valid.**

For more information, see "Java Programming language samples" on page 77.

#### *Review Managed System Entities*

The Managed System security data is referred to in SailPoint IdentityIQ as the Managed System entities. SailPoint IdentityIQ retrieves and manipulates the Managed System security data by getting, adding, updating, and deleting these entities.

Before writing the Connector, you must be familiar with the way security data is represented in the Managed System, and how to retrieve and manipulate it.

SailPoint IdentityIQ entities include:

- **Account** - A log-in Account in a Managed System
- **Managed System Administrator** - An Account with administrator privileges in the Managed System. The credentials for this Account must enable the performance of security-related actions in the Managed System and/or the retrieval of security-related data from the Managed System (according to the functionality required from the Connector).
- **Group** - A Group of Accounts in a Managed System. When Accounts are connected to a Group, they have access to all the resources that can be accessed by that Group.
- **Connection** - A single connection between an Account and a Group (an Account that belongs to a specific Group).
- **Container** - A container is a node in an administrative hierarchical structure (in a Managed System that supports the Container entity) used to manage access rights to information resources within an organization. This hierarchical structure is usually referred to as a Container structure, but may sometimes be called a Directory tree. A Container may hold other Containers and/or other entities such as Accounts or Groups.

  See “Function types” on page 34 for more information on Containers.
- **Resources** represent entity types such as Databases, Tokens, Directories, Files, Shares, and so on that the customer wishes to manage.
- **ResourceACL** represent the list of Access Control Entities that will have the specified access for the resources.

### *Write an independent Proof-of-Concept (POC) (optional)*

Before you begin to design your Connector, SailPoint Technologies recommends that you write and debug an independent sample application that can interact with your Managed System. This sample application should be capable of accessing your Managed System and performing all functionality that will be required of your Connector. Write the sample in the programming language in which you plan to develop your Connector.

## Design

The design stage consists of:

### Map the Managed System Entities to Connector Entities

In order to manage the security environment in an organization, you must determine exactly which security data you wish to manage and how this data is represented in SailPoint IdentityIQ.

You can manage the following Connector entities:

- Managed System Properties
- Accounts
- Groups
- Connections between Accounts and Groups

Each of these Connector entities must be mapped to Managed System entities.

## Map the Managed System Entities parameters to SailPoint IdentityIQ Keywords

Identify the data of every Managed System entity that will be managed by the Connector. Separate the data into fields, and identify the data type of each field (string, integer, or date-time). These data fields will be mapped to Keywords that are transferred between IdentityIQ and the Connector.

For more details, see the ‘Keywords’ section of the “IdentityIQ Connector Factory Development” chapter, in the *SailPoint IdentityIQ Connector Factory User Guide*.

## Identify parameters required by the Connector

Identify any special parameters that will be required in order for the Connector to interact with the Managed System (for example, the port number of the Managed System Oracle Application Server or the location of certain data files). These parameters will be added during the development of the connector.

For more details, see the ‘Configuration Parameters’ section of the “IdentityIQ Connector Factory Development” chapter, in the *SailPoint IdentityIQ Connector Factory User Guide*.

## Analyze the functionality required of the Connector

The API of your Managed System may provide the following entity management functions:

- Functions that enable SailPoint IdentityIQ to administer security in the Managed System

  These functions are required in order to update Managed System data (for example, to create a new Account, to update Account or Group parameters, or to implement password changes). These functions are referred to as Set functions (see “Implementing Set functions” on page 55).
- Functions that enable SailPoint IdentityIQ to retrieve information from the Managed System

  SailPoint IdentityIQ provides functions that retrieve data from the Managed System during data download or after Managed System entity updates, and send this data to IdentityIQ. These functions are referred to as Get functions (see “Implementing Get functions” on page 35).

### *Set functions*

Set functions are SailPoint IdentityIQ functions that insert new Managed System data, or modify or delete existing data.

Data is passed to the Set functions in hash tables that contain context and operation data, as well as entity identification and Keyword data. The entire operation must be performed in a single call to the function. The function’s return code indicates the result of the operation.

New entities are added by Add transactions. Add transactions receive all of the details that were specified for the new entity to be added.

Existing entities are modified by Update transactions. Update transactions receive only the data that was changed for the entity.

Existing entities are deleted by Delete transactions. Delete transactions receive no entity fields or data, other than entity identification.

For more information, see “Set functions” on page 54.

### *Get functions*

Get functions are SailPoint IdentityIQ functions that retrieve Managed System data. They are activated either for a specific object, or for all objects that match a set of criteria.

Data retrieved from Get functions contains entity identification and Keyword data.

A Get function can retrieve the details of either a single entity or of multiple entities, as specified by the scope of the retrieval (all, specific, or prefix). The operation may be performed either in a single call to the function, or in multiple, sequential calls. The function's return code indicates whether more data is pending and the result of the operation.

For more information, see “Get functions” on page 35.

# Chapter 4: The Connector API

When you develop a new Connector, you implement functions in order to access a Managed System. These functions are described in this chapter.

The following topics are discussed:

## Overview

The types of Connector API functions that you should develop in order to access the Managed System are:

- **Flow control functions**: These functions are called upon for Managed System initialization and termination, Connector initialization and termination, session initialization and termination, and to inform the Connector whether the Managed System is active.
- **Get functions**: These functions are used to retrieve information regarding entities (Accounts, Groups, and Account-Group Connections) from the Managed System, as well as information regarding the Managed System itself.
- **Query functions**: These functions are SailPoint IdentityIQ functions that check specific Managed System data.
- **Set functions**: These functions are used to add, update, and delete information regarding entities (Accounts, Groups, and Account-Group Connections), as well as information regarding the Managed System itself, on the Managed System.

You should implement only those Connector functions that are required by your Connector type and the entities that you want to manage. For example, do not implement Set functions for an Audit Connector type.

# Input tables for all functions

All functions receive the following tables as input:

- **XSA_ContextHash** - The context table contains the context variables that the IdentityIQ Connector Factory Framework maintains for the Connector. These variables are automatically switched whenever the context changes. The tables in the function descriptions describe which context scopes are available for each entity. For more information, see "Maintaining the context" on page 70.
- **XSA_OperationHash** - The Operation table contains miscellaneous operation attributes. These attributes are the same for all SailPoint IdentityIQ functions, although their values differ.

Table 1—Common Keywords in the XSA_OperationHash table

| Keyword | Description |
|---|---|
| XSA_ACTION | Name of the action being performed (GET, ADD, UPDATE, DELETE, QUERY) |
| XSA_ENTITY | The entity being handled (SYSTEM_PROPERTIES, ACCOUNT, GROUP, CONNECTION, RESOURCE, RESOURCEACL, CONTAINER) |
| XSA_FUNCTION | The script/function currently running |
| XSA_PROCESS | The name of the current process |
| XSA_CONNECTOR | The name of the current Connector |
| XSA_SYSTEM | The name of the Managed System |
| XSA_WORK_DIR | The path of the work directory (for example, **C:\Program Files\SailPoint Technologies\IdentityIQ\ConnectorManager\Instance_1\Connector\ConnectorNAME\work\**) |
| XSA_TRANSACTION_TYPE | The type of transaction sent from the IdentityIQ.<br><br>For a list of possible values, see Table 2—XSA_TRANSACTION_TYPE possible values on page 18. |
| Additional Keywords that are specific to Get and Set functions | |

In addition to the tables that are common to all functions, there are also specific tables for Get and Set functions; these are described for each function.

Table 2—XSA_TRANSACTION_TYPE possible values (Sheet 1 of 3)

| Value | Description |
|---|---|
| TRANSACTION_INTERCEPT_USER | Account Interception<br><br>(not yet supported) |
| TRANSACTION_INTERCEPT_GROUP | Group Interception<br><br>(not yet supported) |
| TRANSACTION_INTERCEPT_CONNECTION | Connection Interception<br><br>(not yet supported) |

Table 2—XSA_TRANSACTION_TYPE possible values (Sheet 2 of 3)

| Value | Description |
|---|---|
| TRANSACTION_INTERCEPT_PASSWORD | Account Password Interception<br><br>(not yet supported) |
| TRANSACTION_INTERCEPT_USER_REVOKE | Account Revoke Interception<br><br>(not yet supported) |
| TRANSACTION_INTERCEPT_MANAGED_SYSTEM | Managed System keywords Interception<br><br>(not yet supported) |
| TRANSACTION_INTERCEPT_MANAGED_SYSTEM_STATUS | Managed System StatusInterception<br><br>(not yet supported) |
| TRANSACTION_INTERCEPT_EVENT | LOG Event Interception<br><br>(not yet supported) |
| TRANSACTION_GET_USER | Get Account Transaction |
| TRANSACTION_GET_GROUP | Get Group Transaction |
| TRANSACTION_GET_CONNECTION | Get Connection Transaction |
| TRANSACTION_GET_MANAGED_SYSTEM_PARAMETERS | Get Managed System parameters Transaction |
| TRANSACTION_UPDATE_MANAGED_SYSTEM_PARAMETERS | Update Managed System parameters Transaction |
| TRANSACTION_ADD_ADMIN | Add administrator Transaction |
| TRANSACTION_DELETE_ADMIN | Delete administrator Transaction |
| TRANSACTION_UPDATE_ADMIN | Update administrator Transaction |
| TRANSACTION_MOVE_USER | Move Account Transaction<br><br>(not yet supported) |
| TRANSACTION_RENAME_USER | Rename Account Transaction<br><br>(not yet supported) |
| TRANSACTION_ADD_USER | Add Account Transaction |
| TRANSACTION_DELETE_USER | Delete Account Transaction |
| TRANSACTION_UPDATE_USER | Update Account Transaction |
| TRANSACTION_UPDATE_PASSWORD | Update Account password Transaction |
| TRANSACTION_REVOKE_USER | Revoke / Restore Account Transaction |
| TRANSACTION_MOVE_GROUP | Move Group Transaction<br><br>(not yet supported) |
| TRANSACTION_RENAME_GROUP | Rename Group Transaction<br><br>(not yet supported) |

Table 2—XSA_TRANSACTION_TYPE possible values (Sheet 3 of 3)

| Value | Description |
|---|---|
| TRANSACTION_ADD_GROUP | Add Group Transaction |
| TRANSACTION_DELETE_GROUP | Delete Group Transaction |
| TRANSACTION_UPDATE_GROUP | Update Group Transaction |
| TRANSACTION_ADD_CONNECTION | Add Connection Transaction |
| TRANSACTION_DELETE_CONNECTION | Delete Connection Transaction |
| TRANSACTION_UPDATE_CONNECTION | Update Connection Transaction |

# Flow control functions

Table 3—Flow control functions

| Function Group | Function | Description |
|---|---|---|
| Connector loading/unloading | ConnectorInit | Initialize the Connector environment |
| | ConnectorTerm | Cleans up and terminates the Connector environment |
| | SystemActiveQuery | Verifies whether the Managed System is currently active |
| Flow control | ManageSystemInit | Initializes the Connector Managed System environment |
| | ManageSystemTerm | Cleans up and terminates the Connector Managed System environment |
| Session management functions | SessionInit | Logs in an administrator for the first time |
| | SessionTerm | Terminates an administrator session based on the connector session scope |

To better understand the flow control functions, see “IdentityIQ Connector Factory architecture” on page 9.

## Flow function descriptions

This section describes the different flow functions.

### ConnectorInit

**Description**: ConnectorInit is used to initialize the Connector. It is called only once in the Connector lifecycle, immediately after the Connector is loaded by a IdentityIQ.

Use this function to perform all initialization tasks needed for the Connector to work with the one or more Managed Systems with which it interfaces, such as initializing third party libraries, initializing variables, creating dynamic data structures.

Table 4—ConnectorInit Input Tables

| Table | Keyword | Description |
|---|---|---|
| XSA_OperationHash | See also Table 1— Common Keywords in the XSA_OperationHash table on page 18 | |
| XSA_ContextHash | XSA_CONNECTOR_CONTEXT | The connector level context |

Table 5—ConnectorInit XSA_ContextHash Output Table

| Table | Keyword | Description |
|---|---|---|
| XSA_ContextHash | XSA_CONNECTOR_CONTEXT | The connector level context |

Table 6—ConnectorInit Return Codes

| Code | Message |
|---|---|
| XSA_RC_OK (0) | Indicates that the function completed successfully |
| XSA_RC_ERROR (8) | Indicates that an error occurred |

## ConnectorTerm

**Description**: ConnectorTerm performs termination tasks, before the Connector is unloaded.

Use this function to perform termination tasks, such as closing database connections, or deleting dynamic data structures.

Table 7—ConnectorTerm Input Tables

| Table | Keyword | Description |
|---|---|---|
| XSA_OperationHash | See also Table 1 on page -18 | |
| XSA_ContextHash | XSA_CONNECTOR_CONTEXT | The connector level context |

Table 8—ConnectorTerm Return Codes - Output table

| Code | Message |
|---|---|
| XSA_RC_OK (0) | Indicates that the function completed successfully |
| XSA_RC_ERROR (8) | Indicates that an error occurred |

## SystemActiveQuery

**Description**: SystemActiveQuery checks the status of the Managed System, whether it is up or down. IdentityIQ calls this function to verify that the Managed System is currently active. Some Managed Systems can be checked only by activating a login to the Managed System to allow that the Default Administrator name and password reside on the XSA_OperationHash.

However, if the login to the Managed System fails, the error code should be checked. If the error code does not show a problem with the Managed System, the function MUST return XSA_RC_OK, as returning an error here results in a waiting transaction in the IdentityIQ.

For example: For Managed System of type Lotus Notes, any of the following errors may be received:

- Your ID failed authentication check. Access is denied
- Could not open the ID file
- Wrong Password
- Your certificate has expired
- Your User ID has expired

For each of these errors an XSA_RC_OK status should be returned, since the Managed System is available.

**Table 9—SystemActiveQuery Input Tables**

| Table | Keyword | Description |
|---|---|---|
| XSA_OperationHash | XSA_ADMIN_ID | The default administrator |
| | XSA_ADMIN_PASSWORD | The default administrator password |
| | | See also Table 1— Common Keywords in the XSA_OperationHash table on page 18 |
| XSA_ContextHash | XSA_CONNECTOR_CONTEXT | The connector level context |

**Table 10—SystemActiveQuery Output Table**

| Table | Keyword | Description |
|---|---|---|
| XSA_ContextHash | XSA_CONNECTOR_CONTEXT | The connector level context |

**Table 11—SystemActiveQuery Return Codes**

| Code | Message |
|---|---|
| XSA_RC_OK (0) | Indicates that the Managed System is active |
| XSA_RC_NOT_ACTIVE (1) | Indicates that the Managed System is not active |
| XSA_RC_ERROR (8) | Indicates that an error occurred |

## ManageSystemInit

**Description**: ManageSystemInit initializes the Connector Managed System environment. This function is called the first time a transaction is called on a Managed System, and every time the Managed System is changed.

**Table 12—ManageSystemInit Input Tables**

| Table | Keyword | Description |
|---|---|---|
| XSA_OperationHash | See also Table 1— Common Keywords in the XSA_OperationHash table on page 18 | |
| XSA_ContextHash | XSA_CONNECTOR_CONTEXT | The connector level context |

Table 13—ManageSystemInit Output Table

| Table | Keyword | Description |
|---|---|---|
| XSA_ContextHash | XSA_CONNECTOR_CONTEXT | The connector level context |
| | XSA_SYSTEM_CONTEXT | Managed System level context |

Table 14—ManageSystemInit Return Codes

| Code | Message |
|---|---|
| XSA_RC_OK (0) | Indicates that the function completed successfully |
| XSA_RC_ERROR (8) | Indicates that an error occurred |

## ManageSystemTerm

**Description:** ManageSystemTerm cleans up and terminates the Connector Managed System environment.

Table 15—ManageSystemTerm Input Tables

| Table | Keyword | Description |
|---|---|---|
| XSA_OperationHash | See also Table 1— Common Keywords in the XSA_OperationHash table on page 18 | |
| XSA_ContextHash | XSA_CONNECTOR_CONTEXT | The connector level context |
| | XSA_SYSTEM_CONTEXT | The Managed System level context |

Table 16—ManageSystemTerm XSA_ContextHash Output Table

| Table | Keyword | Description |
|---|---|---|
| XSA_ContextHash | XSA_CONNECTOR_CONTEXT | The connector level context |

Table 17—ManageSystemTerm Return Codes

| Code | Message |
|---|---|
| XSA_RC_OK (0) | Indicates that the function completed successfully |
| XSA_RC_ERROR (8) | Indicates that an error occurred |

## TransactionStart

**Description**: The TransactionStart function is called at the start of every transaction. The developer can look at the XSA_SERVICE_TYPE and XSA_TRANSACTION_TYPE found at the XSA_OperationHash to understand which transaction is being performed and initialize the transaction environment accordingly.

Table 18—TransactionStart Input Tables

| Table | Keyword | Description |
|---|---|---|
| XSA_OperationHash | See also Table 1— Common Keywords in the XSA_OperationHash table on page 18 | |
| XSA_ContextHash | XSA_CONNECTOR_CONTEXT | The connector level context |
| | XSA_SYSTEM_CONTEXT | The Managed System level context |

Table 19—TransactionStart Output Table

| Table | Keyword | Description |
|---|---|---|
| XSA_ContextHash | XSA_CONNECTOR_CONTEXT | The connector level context |
| | XSA_SYSTEM_CONTEXT | The Managed System level context |
| | XSA_TRANSACTION_CONTEXT | The Transaction level context |

Table 20—TransactionStart Return Codes

| Code | Message |
|---|---|
| XSA_RC_OK (0) | Indicates that the function completed successfully |
| XSA_RC_ERROR (8) | Indicates that an error occurred |

An example of a possible use is for an IdentityIQ Full Download, when Get functions for all entities are being called. For some Managed Systems the information from a Get Function of one entity can be used by the Get Function of another entity. For example, Groups information from GroupGet and Accounts information from AccountGet can be used for ConnectionGet.

The developer can use the XSA_OperationHash key XSA_SERVICE_TYPE to check for a download or Global Sync situation. Global data structures can then be initialized to be alive and used by all Get functions and be deleted only at transaction end.

## TransactionEnd

**Description**: The TransactionEnd function is called at the end of every transaction, and can be used to terminate the transaction environment.

Table 21—TransactionEnd Input Tables

| Table | Keyword | Description |
|---|---|---|
| XSA_OperationHash | See also Table 1— Common Keywords in the XSA_OperationHash table on page 18 | |
| XSA_ContextHash | XSA_CONNECTOR_CONTEXT | The connector level context |
| | XSA_SYSTEM_CONTEXT | The Managed System level context |
| | XSA_TRANSACTION_CONTEXT | The Transaction level context |

Table 22—TransactionEnd Output Table

| Table | Keyword | Description |
|---|---|---|
| XSA_ContextHash | XSA_CONNECTOR_CONTEXT | The connector level context |
| | XSA_SYSTEM_CONTEXT | The Managed System level context |

Table 23—TransactionEnd Return Codes

| Code | Message |
|---|---|
| XSA_RC_OK (0) | Indicates that the function completed successfully |
| XSA_RC_ERROR (8) | Indicates that an error occurred |

## The Connector Session Mechanism

An administrator session is the period between the SessionInit function and the SessionTerm function. During this period, every connector function called, that uses this administrator for its action will have the correct administrator and password on the **XSA_OperationHash**, and the correct Session Context on the **XSA_ContextHash**.

For example, when an **AddAccount** transaction is sent from the IdentityIQ there are actually two actions sent to the connector:

- AccountAdd - to add the new account
- AccountGet - to double check and see that the new account exists (hotpath)

The connector AccountAdd function will be called with the Managed System Administrator name and password on the **XSA_OperationHash** and its appropriate context on the **XSA_ContextHas**h at key XSA_SESSION_CONTEXT

The connector AccountGet function will be called with the Default Administrator name and password on the **XSA_OperationHash** and its appropriate context on the **XSA_ContextHash** at key XSA_SESSION_CONTEXT.

## Session Scope

There are two possible options to use with the SessionInit and SessionTerm mechanism. The option chosen by the developer defines the **Session Scope**.

The possible choices for the Session Scope are:

- **Transaction Scope**: Sessions are opened and closed between the TransactionStart and TransactionEnd functions. After TransactionStart, SessionInit will be called once for the Managed System Administrator used for that transaction, and once for the Default Administrator.

  Immediately before TransactionEnd, SessionTerm will be called for both the open sessions.

  The disadvantage of this method is that there is a large overhead involved in connecting to the database for each and every transaction. The advantage of the method is that the sessions are only open for a short time.
- **Managed System Scope**: - Sessions are opened and closed between the ManageSystemInit and ManageSystemTerm functions. After ManageSystemInit the first time that an administrator is used to perform actions on the Managed System, SessionInit will be called once for that Administrator.

  Immediately before MangeSystemTerm, SessionTerm will be called for all open sessions.

  The disadvantage of this method is that sessions may be open for a very long time. The advantage of the method is that there are less calls to the database.

Because of the possibility of sessions being open for a long time, it can help to verify that the Connection/Login handle you got at SessionInit is active, and if it is not, then to reconnect and get the handle/connection again.

**Note: In a Managed System Administered by Multiple IIQ Administrators each with a different Managed System Administrator you may have more than two sessions opened - one for the default Administrator and a other individual sessions for each Managed System Administrator.**

## SessionInit

**Description**: SessionInit initializes an administrator login session to the Managed System. This function is called according to the Session Scope chosen by the developer:

- Managed System scope: SessionInit will be called the first time that an administrator is used to perform actions on the Managed System.
- Transaction scope: SessionInit will be called at the start of the transaction after TransactionInit and before the actual set/get functions are called.

Table 24—SessionInit Input Tables

| Table | Keyword | Description |
|---|---|---|
| XSA_OperationHash | XSA_ADMIN_ID | The administrator performing the action |
| | XSA_ADMIN_PASSWORD | The password of the administrator |
| | See also Table 1— Common Keywords in the XSA_OperationHash table on page 18 | |
| XSA_ContextHash | XSA_CONNECTOR_CONTEXT | The connector level context |
| | XSA_SYSTEM_CONTEXT | The system context level |
| | XSA_TRANSACTION_CONTEXT | The transaction context level |

Table 25—SessionInit Output Table

| Table | Keyword | Description |
|---|---|---|
| XSA_ContextHash | XSA_CONNECTOR_CONTEXT | The connector level context |
| | XSA_SESSION_CONTEXT | The session level context |
| | XSA_SYSTEM_CONTEXT | The system context level |
| | XSA_TRANSACTION_CONTEXT | The transaction context level |

Table 26—SessionInit Return Codes

| Code | Message |
|---|---|
| XSA_RC_OK (0) | Indicates that changes were successfully applied |
| XSA_RC_ERROR (8) | Indicates that an error occurred |

### *Using SessionInit*

You can use SessionInit to:

- Initiate a login session to the Managed System

  You can implement login/logout to the Managed System for each API function. Another method, which is more efficient, is to use SessionInit to initiate a login session to the Managed System and then logout on SessionTerm. This eliminates the need to login/logout for each API function.
- Keep information in the Session Context

  If you put information in the Session Context of a particular administrator, for a specific Managed System, Session Context will be available for any of the Connector functions used to perform an action with that administrator for the Managed System. Therefore, you can use it to hold the login handle/context or any other information you find valuable.

For example, for TDB_JAVA: SessionInit is used to login to the System and to put the database path on the Session Context.

For additional information on Managed System architecture, see "IdentityIQ Connector Factory architecture" on page 9.

## SessionTerm

**Description**: SessionTerm is called to terminate (log out) an administrator login session. This function is called once for each administrator session, according to the Session Scope.

- Managed System scope: – SessionTerm will be called immediately before ManageSystemTerm
- Transaction scope: SessionTerm will be called immediately before TransactionEnd

Table 27—SessionTerm Input Tables

| Table | Keyword | Description |
|---|---|---|
| XSA_OperationHash | XSA_ADMIN_ID | The administrator performing the action |
| | XSA_ADMIN_PASSWORD | The password of the administrator |
| | See also Table 1— Common Keywords in the XSA_OperationHash table on page 18 | |
| XSA_ContextHash | XSA_CONNECTOR_CONTEXT | The connector level context |
| | XSA_SESSION_CONTEXT | The session level context |
| | XSA_SYSTEM_CONTEXT | The system level context |

Table 28—SessionTerm Output Table

| Table | Keyword | Description |
|---|---|---|
| XSA_ContextHash | XSA_CONNECTOR_CONTEXT | The connector level context |
| | XSA_SYSTEM_CONTEXT | The system level context |

Table 29—SessionTerm Return Codes

| Code | Message |
|---|---|
| XSA_RC_OK (0) | Indicates that the function completed successfully |
| XSA_RC_ERROR (8) | Indicates that an error occurred |

For additional information on Managed System architecture, see “IdentityIQ Connector Factory architecture” on page 9.

## Flow function sequences

This section describes the flow of function calls made by IdentityIQ when it processes a transaction using the IdentityIQ Connector Factory API. The term **Transaction** is used to describe the sequence of events occurring when:

- IdentityIQ issues a request to Connector to modify Managed System data
- IdentityIQ issues a request to Connector to retrieve data from the Managed System

Each transaction is identified by an SIID (Service Instance ID). The SIID uniquely identifies a transaction across all platforms and Managed Systems managed by IdentityIQ.

## Possible function flows when Session Scope is Transaction

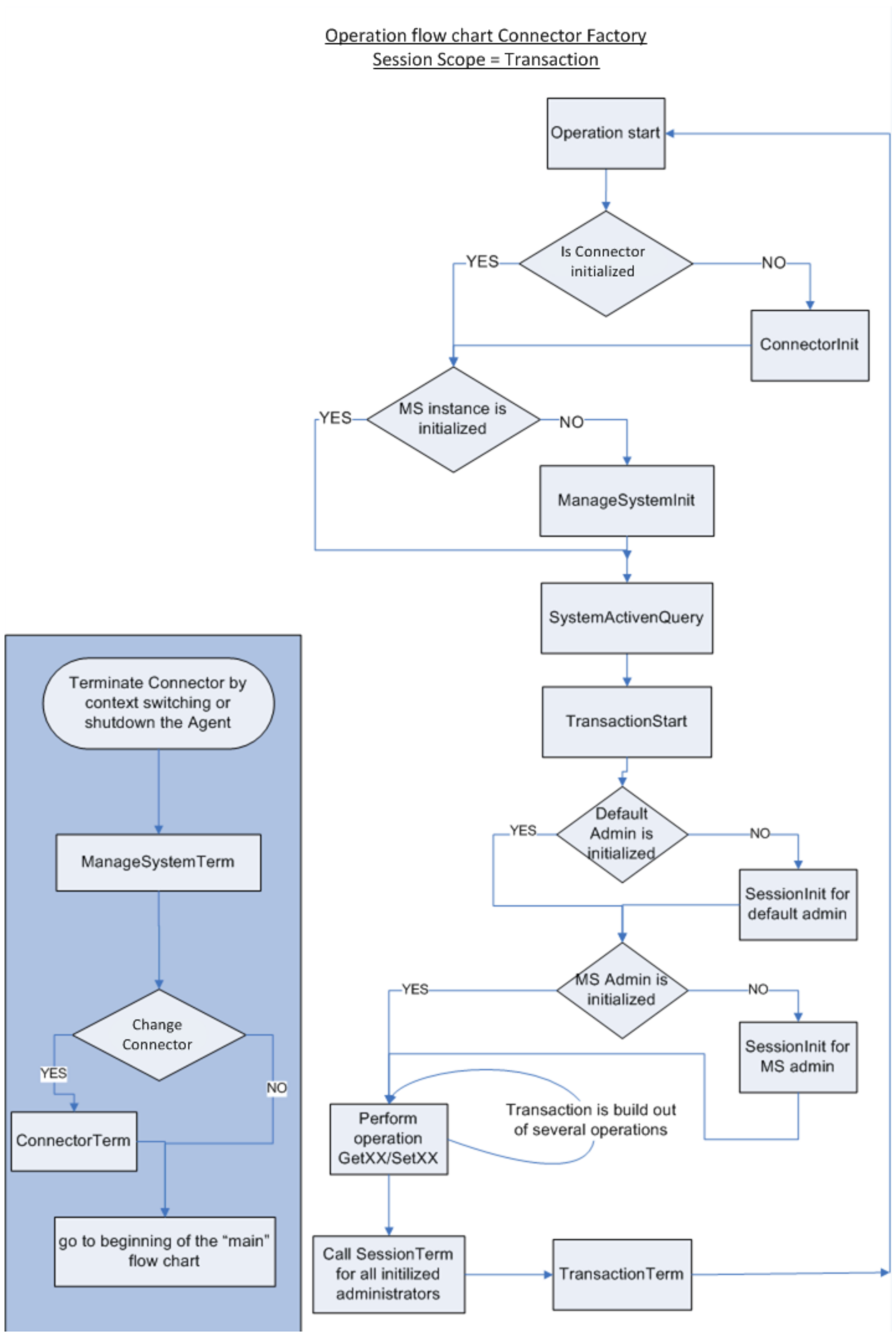
Operation flow chart Connector Factory
Session Scope = Transaction
Operation start
Is Connector initialized
YES
NO
ConnectorInit
MS instance is initialized
YES
NO
ManageSystemInit
SystemActivenQuery
TransactionStart
Default Admin is initialized
YES
NO
SessionInit for default admin
MS Admin is initialized
YES
NO
SessionInit for MS admin
Perform operation GetXX/SetXX
Transaction is build out of several operations
Call SessionTerm for all initilized administrators
TransactionTerm
Terminate Connector by context switching or shutdown the Agent
ManageSystemTerm
Change Connector
YES
NO
ConnectorTerm
go to beginning of the "main" flow chart

Figure 2—Session Scope is transaction flow

**Flow control functions**

***For the first transaction sent from IdentityIQ to Connector:***

- Connection management

  ConnectorLoad - to load the Connector

  ConnectorInit - to initialize the Connector

  ManageSystemInit to initialize the Connector Managed System
- Transaction processing

  SystemActiveQuery

  TransactionStart – to initialize the Transaction

  SessionInit -- Called once or twice with the Administrators used for transaction.

  Set/Get/Query functions – could be many calls

  SessionTerm – Called for all Administrators used

  TransactionEnd

***Transaction Sent for the same Managed System already initialized***

- Transaction processing

  SystemActiveQuery

  TransactionStart – to initialize the Transaction

  SessionInit -- Called once or twice with the Administrators used for transaction.

  Set/Get/Query functions – could be many calls

  SessionTerm – Called for all Administrators used

  TransactionEnd

***Transaction sent from IdentityIQ to Connector for a different Managed System in the same Connector after the Connector was already loaded***

- Connection management

  ManagedSystemTerm – to terminate the current Connector Managed System environment

  ManageSystemInit - to initialize the new Connector Managed System environment
- Transaction processing

  SystemActiveQuery

  TransactionStart – to initialize the Transaction

  SessionInit -- Called once or twice with the Administrators used for transaction.

  Set/Get/Query functions – could be many calls

  SessionTerm – Called for all Administrators used

  TransactionEnd

***Transaction sent from the IdentityIQ to the Connector for a different Managed System with a different Connector***

- Connection management

  ManageSystemTerm - to terminate the current Connector Managed System environment

  ConnectorTerm - to terminate the current Connector

  ConnectorLoad - to load the new Connector

  ConnectorInit - to initialize the new Connector

  ManageSystemInit - to initialize the new Connector Managed System Environment

- Transaction processing
  SystemActiveQuery
  TransactionStart – to initialize the Transaction
  SessionInit -- Called once or twice with the Administrators used for transaction.
  Set/Get/Query functions – could be many calls
  SessionTerm – Called for all Administrators used
  TransactionEnd

## Possible function flows when Session Scope is Managed System

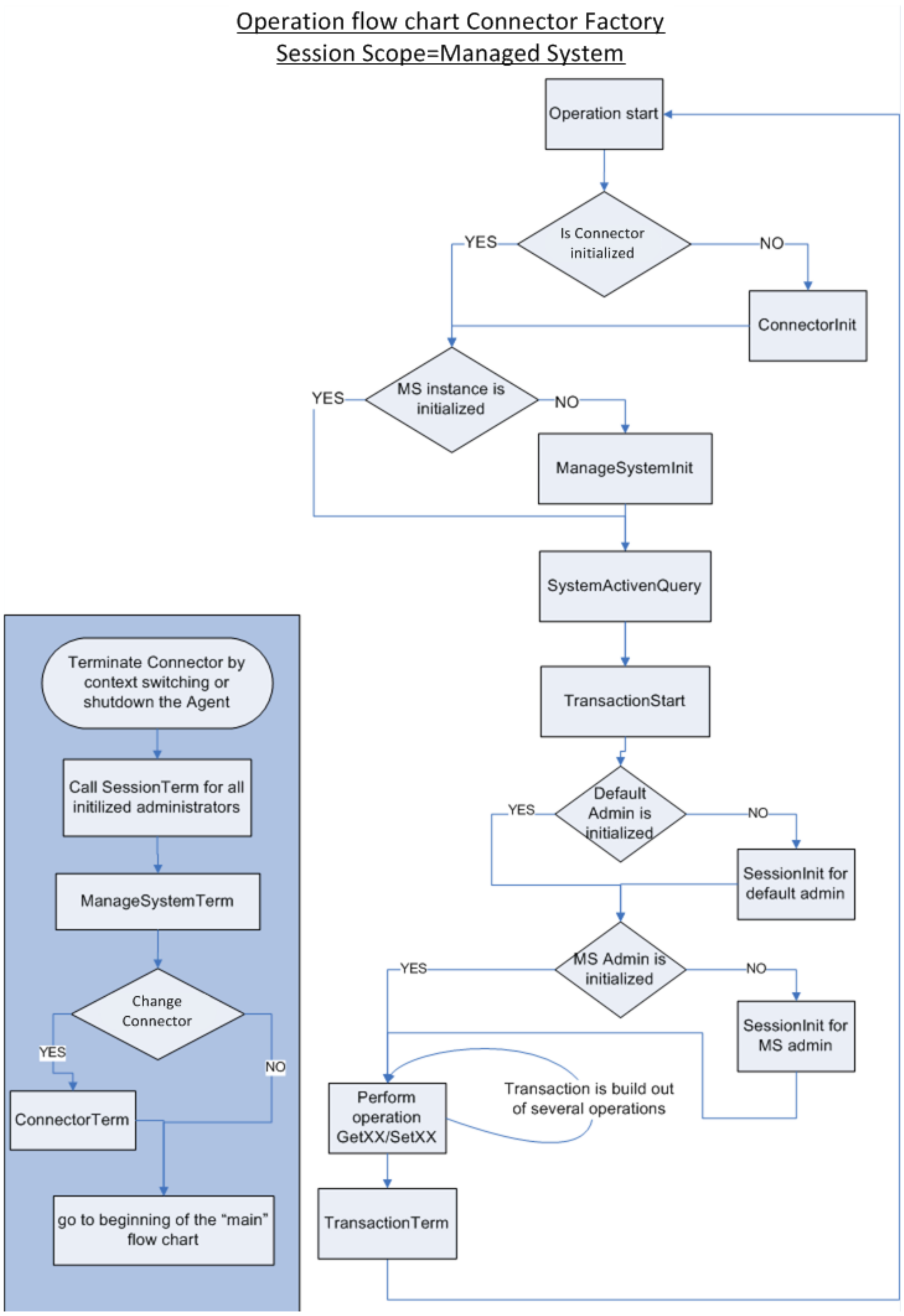
Operation flow chart Connector Factory
Session Scope=Managed System
Operation start
Is Connector initialized
YES
NO
ConnectorInit
MS instance is initialized
YES
NO
ManageSystemInit
SystemActivenQuery
TransactionStart
Default Admin is initialized
YES
NO
SessionInit for default admin
MS Admin is initialized
YES
NO
SessionInit for MS admin
Perform operation GetXX/SetXX
Transaction is build out of several operations
TransactionTerm
Terminate Connector by context switching or shutdown the Agent
Call SessionTerm for all initilized administrators
ManageSystemTerm
Change Connector
YES
NO
ConnectorTerm
go to beginning of the "main" flow chart

Figure 3—Session scope is Managed System flow

### *For the first transaction sent from IdentityIQ to Connector:*

- Connection management

  ConnectorLoad - to load the Connector

  ConnectorInit - to initialize the Connector

  ManageSystemInit to initialize the Connector Managed System
- Transaction processing

  SystemActiveQuery

  TransactionStart – to initialize the Transaction

  SessionInit - Called once or twice with the Administrators used for the transaction.

  Set/Get/Query functions – could be many calls

  TransactionEnd

### *Transaction Sent for the same Managed System already initialized*

- Transaction processing

  SystemActiveQuery

  TransactionStart – to initialize the Transaction

  SessionInit - Called only if the Transaction Managed System administrator was not initialized in the previous SessionInit calls.

  Set/Get/Query functions – could be many calls

  TransactionEnd

### *Transaction sent from the IdentityIQ to Connector for a different Managed System in the same Connector after the Connector was already loaded*

- Connection management

  SessionTerm – to end all open Administrators Sessions.

  ManagedSystemTerm – to terminate the current Connector Managed System environment

  ManageSystemInit - to initialize the new Connector Managed System environment
- Transaction processing

  SystemActiveQuery

  TransactionStart – to initialize the Transaction

  SessionInit - Called once or twice with the Administrators used for transaction.

  Set/Get/Query functions – could be many calls

  TransactionEnd

### *Transaction sent from the IdentityIQ to the Connector for a different Managed System with a different Connector*

- Connection management

  SessionTerm – to end all open Administrators Sessions.

  ManageSystemTerm - to terminate the current Connector Managed System environment

  ConnectorTerm - to terminate the current Connector

  ConnectorLoad - to load the new Connector

  ConnectorInit - to initialize the new Connector

  ManageSystemInit - to initialize the new Connector Managed System Environment

- Transaction processing

  SystemActiveQuery

  TransactionStart – to initialize the Transaction

  SessionInit - Called once or twice with the Administrators used for transaction.

  Set/Get/Query functions – could be many calls

  TransactionEnd

# Entity management functions

This section describes the different entity management functions.

## Function types

The implementation of the Managed System interface should provide the following entity management function types:

- Get functions that enable SailPoint IdentityIQ to request information from the Managed System.

  These are functions that provide get of entity information from the Managed System. The Entities can be of type account, group, connection, container, resource, resource ACL and the Managed System Settings.

  An overview of Get functions is provided below. See “Get functions” on page 35.
- Set functions that enable SailPoint IdentityIQ to administer security in the Managed System.

  These are functions that provide management of entity information on the Managed System. The Entities can be of type account, group, connection, container, resource, resource ACL and the Managed System Settings.

  An overview of Set functions is provided in “Set functions” on page 54.

The Entity Management Functions perform activities for the following entities:

- Accounts
- Groups
- Account to Group Connections
- Managed System Properties
- Containers
- Resources
- Resource ACLs

**Note: In the IIQ, Containers always have one Root parent container which is the root of the whole container tree. The IdentityIQ Connector Factory framework automatically creates the root container with the name of the Managed System configured in IIQ.**

**This Root container is seen only in the IIQ, the IdentityIQ Connector Factory framework is responsible to remove or concatenate it to the entity names sent to the connector or received from the connector.**

**For example, an Account name for a connector programmer of Japan/Tokyo/Yuko, will be seen in the IIQ by the TDBJava Managed System as TdbJava/Japan/Tokyo/Yuko**

Table 30— Entity Management Functions summarizes entity function calls in Connector that are used to interact with the Managed System.

Table 30—Entity Management Functions

| Function Group | Function | Description |
|---|---|---|
| Managed System "Set" Functions | SystemPropertiesSet | Sets Managed System properties |
| | AccountAdd | Creates a new Account on the Managed System |
| | AccountDelete | Deletes an Account from the Managed System |
| | AccountUpdate | Modifies an existing Account on the Managed System, including the following functionality:<br>Revoke Account<br>Restore Account<br>Update password |
| | GroupAdd | Creates a new Group on the Managed System |
| | GroupDelete | Deletes a Group from the Managed System |
| | GroupUpdate | Updates the attributes of an existing Group |
| | ConnectionAdd | Creates a new Connection between an Account and a Group |
| | ConnectionDelete | Deletes the Connection between an Account and a Group |
| | ConnectionUpdate | Modifies the attributes of an existing Connection between an Account and a Group |
| Managed System "Get" Functions | SystemPropertiesGet | Retrieves Managed System properties |
| | AccountGet | Retrieves Account data |
| | GroupGet | Retrieves Group data |
| | ConnectionGet | Retrieves Account to Group Connections data |
| Managed System Query functions | VerifyAdminPassword | Verify Password for an Administrator on the Managed System |
| | VerifyAccountPassword | Verify Password for an Account on the Managed System |

## Get functions

This section describes the Get functionality of the Connector Factory.

### Implementing Get functions

Get functions are Connector functions that retrieve Managed System data. They are activated for a specific object, or for all objects that match a set of criteria.

Get functions fill tables with data retrieved from the Managed System. These tables contain context and operation data, as well as entity filter and Keyword data.

A Get function can retrieve the details of a single entity or of multiple entities. The scope of the retrieval (all entities, a specific entity, or entities that begin with a specified prefix) is defined by the XSA_FILTER_TYPE Keyword, which is passed to the function in the XSA_OperationHash table.

## Entity retrieval

The purpose of all Get actions is to fetch one or more entities. The data from those entities is either sent on to IIQ, or used by the standard offline interceptor to identify changes that were made directly to the Managed System. The entity fetched can be an Account, a Group, or a Connection.

An entity is composed of Keywords. These Keywords are found in the XSA_GetEntityHash hash table, which serves as input to all the Connector functions. The Connector code must be capable of correctly filling the values in this hash table. The Connector code does not check the validity of the values, nor does it verify that all mandatory fields are filled. Those actions are performed by IIQ Gateway. Examples of such errors can be seen in the IdentityIQ alerts list.

The Connector code should fill the values for one entity, and then call the Framework service XSA_WriteEntity.

The XSA_WriteEntity service is called according to the value of the XSA_FILTER_TYPE entry in the XSA_OperationHash hash table, as shown in Table 31— XSA_FILTER_TYPE values and actions.

Table 31—XSA_FILTER_TYPE values and actions

| Value | Action |
|---|---|
| SINGLE | The Connector code should call the XSA_WriteEntity service only once, if the entity exists.<br><br>If the entity does not exist, the XSA_WriteEntity service should not be called. |
| MANY, SUBTREE or PREFIX | The Connector code can call the service many times, once for each entity. Before each call, the Connector code fills the XSA_GetEntityHash hash table with one entity, and then calls the service. It repeats this process for all entities that match the Get request |
| Other | Not a valid Get action, and any call to the service produces an error. |

If a large number of entities match the request, the Connector code should retrieve a number of them, then return to the Framework with a return code of XSA_RC_MORE. The Framework will process the retrieved entities, then return to the Connector to retrieve more. Use the ACTION context (see "Maintaining the context" on page 70) to keep track of where the previous retrieval left off. (The Get function can use either the context or global variables, in order to remember which entity was the last one returned.) When all of the appropriate entities have been retrieved, the return code should be XSA_RC_OK.

The maximum number of entities retrieved each time should be determined by the size of the entity (Keyword names and values). Note that each call to the service stores the entity in memory until the Connector code has finished running, and the memory required for doing so should not exceed the computer capacity. You must also consider the capacity of the Managed System, that is how many entities it allows to be fetched at a time.

## Handling Get requests for Accounts, Groups, and Connections

### *The Stage of a Get Action*

Get functions receive a parameter that indicates whether the current call is the first call, a continuation call, or the last call to the function.

This parameter is called XSA_GET_STAGE, and is contained in the XSA_OperationHash hash table. Valid values for this parameter are START, NEXT , and END.

- START - Indicates this is the first call to the Get function. The function should perform initialization tasks (if required), and may also return entities. If the function terminates with RC_OK while in START stage, there will not be another call in END stage.
- NEXT - Indicates this is not the first call to the Get function, and that the previous call to the function had the return code XSA_RC_MORE. The function should return entities.
- END - Indicates that the function should perform termination tasks (if required). The function must not return entities while in END stage. The Get function is called in the END stage only if the previous call to the function was in NEXT stage and ended with RC_OK .

### *How can I determine the current stage of a Get action?*

The Get stage can be retrieved from the Keyword named XSA_GET_STAGE contained `in the` XSA_OperationHash table.

The following is an example of mode retrieval using Java:

**getStage = (String)XSA_OperationHash.get("XSA_GET_STAGE");**

### *Get Functions Entity Search ID*

Get functions receive an ID key that describes the ID to be retrieved. The information is received in the XSA_GetSearchHash table and contains values according to the get mode.

The possible keys in the GetSearchHash are:

- XSA_ACCOUNT_NAME
- XSA_GROUP_NAME

### *The request mode of a Get action*

Get functions can be called in one of the following modes: SINGLE, MANY, SUBTREE, or PREFIX. The request mode determines the scope of the retrieval.

- SINGLE - Retrieves one entity from the Managed System. The unique ID of the entity to be retrieved is received as input.

  For Accounts and Groups the ID Key of the entity name is XSA_*entityName*_NAME.

  For Connections, the ID keys describing a connection are:

  - XSA_ACCOUNT_NAME
  - XSA_GROUP_NAME

- MANY – For Accounts and Groups:
    - If no ID key is received the Get function should retrieve all the entities from the Managed System
    - If an ID key was received this is a restart download transaction and the Entity ID is the last entity received at the previous failed download. See "Transaction Server" on page 109.
    - For Connections, two ID keys are received; one for the Account and one for the Group:
    - If both ID keys are empty, the Get function should retrieve all existing Connections from the Managed System
    - If the Account ID key is received and the Group ID key is empty, the function should retrieve and return all the Connections for the given Account.
    - If the Account ID key is empty, and the Group ID key contains a value, the Get function should retrieve and return all the Connections for the given Group.
    - If both keys are full, this is a restart download transaction and the Group and Account Ids represent the last connection retrieved at the previous failed download
- PREFIX - For Accounts only. Indicates that the Get function should retrieve Accounts that match the given prefix. The retrieved Account will be found on ID key XSA_ACCOUNT_NAME.

### *How can I determine the request mode of a Get action?*

The Get request mode can be retrieved from the Keyword named XSA_FILTER_TYPE in the XSA_OperationHash table.

The following is an example of mode retrieval using Java:

**filterType =**

**(String)XSA_OperationHash.get("XSA_FILTER_TYPE");**

### *How Can I Retrieve the Entity ID?*

The relevant Entity ID is retrieved according to the entity type and the mode of the get function. For example, for a connector that implements Containers Management, the ID for a get action in a SINGLE,MANY or PREFIX mode is found at ID key XSA_ACCOUNT_NAME.

The following is an example of ID retrieval of an Account using Java:

**accountName = (String)**

**XSA_GetSearchHash.get("XSA_ACCOUNT_NAME");**

### *How can I retrieve an Entity?*

Assuming that the variables accountName, adminStatus, and accountComment contain the relevant entity data extracted from the Managed System, the following Java example returns an entity:

```
XSA_GetEntityHash.put("XSA_ACCOUNT_NAME", accountName);
XSA_GetEntityHash.put("XSA_ADMIN_STATUS", adminStatus);
.
.  (other Keywords)
.
XSA_GetEntityHash.put("TDB_JAVA_ACCOUNT_COMMENT", accountComment);
rc = XSA_Framework.XSA_WriteEntity();
```

#### *In a Get function in Single mode, how do I notify the IdentityIQ Connector Factory framework that an entity does not exist?*

If an entity does not exist, the Connector code should not call the service XSA_WriteEntity and return with and XSA_RC_OK status. This notifies the framework that the entity does not exist.

## Get function input tables

In addition to the input described in Table 1— Common Keywords in the XSA_OperationHash table on page 18, all Get functions (SystemPropertiesGet, AccountGet, GroupGet, ConnectionGet, ConntainerGet) receive the following tables:

- XSA_GetEntityHash: This table contains placeholders for all Keywords that must be filled by the Get function. For the specific fields in this table, see the relevant description for each function.
- XSA_GetSearchHash: This table specifies search filter criteria for retrieving entities. If this table is empty, all entities should be retrieved. For the specific filter type, see the relevant description for each function.

## Get functions output tables

All Get functions return entities by repeating the following actions, once for each entity retrieved:

- Fill the XSA_GetEntityHash table with the entity data.
- Call the Framework service XSA_WriteEntity. For more information, see “Framework services” on page 67.

## Typical scenarios for Get actions

### *Getting a single Entity*

If the request was to retrieve one single entity from the Managed System, then the unique ID key of the entity is provided as input.

1. Determine the stage of the Get action. If the stage is START or NEXT, continue to step 2. If the Get stage is END, perform termination tasks, if necessary, and return the function.
2. Extract the ID key to be retrieved from the input hash table.
3. Using this ID key, find its entity and retrieve its data from the Managed System.
4. If an entity is found:
    a. Fill the XSA_GetEntityHash output hash table with the data retrieved from the Managed System.
    b. Call the XSA_WriteEntity function.
5. Return the function with the return code XSA_RC_OK.

### *Getting multiple Entities*

If the request was to retrieve entities that match a filter (prefix), entities in a subtree of a container, or to retrieve all entities, the Get function can return zero or more entities. If the Get function has more than one entity to return, it can do so in the following ways:

**To return all Entities in a single call, perform the following:**

By using the following procedure, the Get function will return the data for all the entities that match the filter in one call. This method is not recommended for retrieving a large `number` of entities – since the entities will reside in the IdentityIQ Connector Factory Framework memory until all are sent to IdentityIQ, this can create memory problems.

1. Determine the stage of the Get action. If the stage is START or NEXT, continue to step 2.. If the Get stage is END, perform termination tasks, if necessary, and return the function.
2. If the request mode is PREFIX, extract the prefix from the input hash table at key XSA_ACCOUNT_NAME.
3. Retrieve the data for the next entity that matches the criteria from the Managed System.
4. Fill the XSA_GetEntityHash output hash table with the data retrieved from the Managed System for the current single entity.
5. Call the XSA_WriteEntity function.
6. Return to step 3., and repeat until the last entity has been retrieved.
7. Return the function with the return code XSA_RC_OK.

**To return a single Entity for each call, perform the following:**

By using the following procedure, the Get function will return data for a single entity each time it is called.

1. Determine the stage of the Get action. If the stage is START or NEXT, continue to step 2.. If the Get stage is END, perform termination tasks, if necessary, and return the function.
2. If the request mode is PREFIX, extract the prefix from the input hash table at key XSA_ACCOUNT_NAME.
3. Retrieve the data for the next entity that matches the criteria from the Managed System.
4. Fill the XSA_GetEntityHash output hash table with the data retrieved from the Managed System for the current single entity.
5. Call the XSA_WriteEntity function.
6. Return the function with the return code XSA_RC_MORE.
7. Return to step 3., and repeat until the last entity has been retrieved. For the last entity, return the function with the return code XSA_RC_OK.

The Get function can use either the context data string or global variables, in order to remember which entity was the last one retrieved.

For more information see “Maintaining the context” on page 70.

**To return one or more Entities for each call, perform the following:**

Using the following procedure, the Get function will return any number of entities each time it is called.

1. Determine the stage of the Get action. If the stage is START or NEXT, continue to step 2.. If the Get stage is END, perform termination tasks, if necessary, and return the function.
2. If the request mode is PREFIX, extract the prefix from the input hash table at key XSA_ACCOUNT_NAME.
3. Retrieve the data for the next entity that matches the criteria from the Managed System.
4. Fill the XSA_GetEntityHash output hash table with the data retrieved from the Managed System for the current single entity.
5. Call the XSA_WriteEntity function.
6. Return to step 3., and repeat until the number you wanted to return was reached.
7. If there are no more entities to retrieve, return the function with the return code XSA_RC_OK. If there are more entities to retrieve, return the function with the return code XSA_RC_MORE.

The Get function can use either the context data string or global variables, in order to remember which entity was the last one returned. For more information see “Maintaining the context” on page 70.

**Note: See the TDB_JAVA sample project Get Function for an example – of returning a number of entities per function call.**

## Get functions reference

This section describes the different operations of Get function.

### AccountGet

**Description**: AccountGet retrieves Account data from the Managed System.

Table 32—AccountGet Input Table (Sheet 1 of 2)

| Table | Keyword | Description |
|---|---|---|
| XSA_OperationHash | XSA_FILTER_TYPE | The type of filter (SINGLE, MANY, PREFIX, or SUBTREE) |
| | XSA_GET_STAGE | The stage of the Get action (START, NEXT, or END) |
| | XSA_ADMIN_ID | The administrator performing the action |
| | XSA_ADMIN_PASSWORD | The password of the administrator |
| | See Table 1— Common Keywords in the XSA_OperationHash table on page 18. | |
| XSA_ContextHash | XSA_CONNECTOR_CONTEXT | The connector level context |
| | XSA_ACTION_CONTEXT | The action level context or placeholder |
| | XSA_SESSION_CONTEXT | The session level context |
| | XSA_SYSTEM_CONTEXT | The Managed System level context |
| | XSA_TRANSACTION_CONTEXT | The transaction level context |
| XSA_GetSearchHash | XSA_ACCOUNT_NAME | Filter entities using this Keyword |
| | XSA_PARENT_CONTAINER | Filter entities using this Keyword |

Table 32—AccountGet Input Table (Sheet 2 of 2)

| Table | Keyword | Description |
|---|---|---|
| XSA_GetEntityHash<br><br>See Table 35—AccountGet EntityHash Keyword Values Table on page 43. | XSA_ACCOUNT_NAME | Placeholder |
| | XSA_PASSWORD_LIFE | Placeholder |
| | XSA_REVOKE_STATUS | Placeholder |
| | XSA_ADMIN_STATUS | Placeholder |
| | XSA_DEF_GROUP | Placeholder |
| | XSA_DEF_GROUP_ACTION | Placeholder |
| | XSA_LOCK_STATUS | Placeholder |
| | User defined Keywords | |

Table 33—AccountGet Output Table

| Table | Keyword | Description |
|---|---|---|
| XSA_ContextHash | XSA_CONNECTOR_CONTEXT | The connector level context |
| | XSA_ACTION_CONTEXT | The action level context or placeholder |
| | XSA_SESSION_CONTEXT | The session level context |
| | XSA_SYSTEM_CONTEXT | The Managed System level context |
| | XSA_TRANSACTION_CONTEXT | The transaction level context |
| XSA_GetEntityHash<br><br>See Table 35—AccountGet EntityHash Keyword Values Table on page 43. | XSA_ACCOUNT_NAME | The Account name |
| | XSA_PASSWORD_LIFE | Whether the Password is temporary or permanent |
| | XSA_REVOKE_STATUS | The Revoke status of the returned Account |
| | XSA_ADMIN_STATUS | The Administrative status of the returned Account |
| | XSA_DEF_GROUP | The Account's default Group |
| | XSA_DEF_GROUP_ACTION | The action to be performed on the old default group connection |
| | XSA_LOCK_STATUS | The Lock status of the returned Account |
| | User defined Keywords | |

Table 34—AccountGet Return Codes (Sheet 1 of 2)

| Code | Message |
|---|---|
| XSA_RC_OK (0) | Indicates that all entities were successfully retrieved |

Table 34—AccountGet Return Codes (Sheet 2 of 2)

| Code | Message |
|---|---|
| XSA_RC_MORE(1) | Indicates that this function should be called again to retrieve additional entities |
| XSA_RC_ERROR (8) | Indicates that an error occurred |

Table 35—AccountGet EntityHash Keyword Values Table (Sheet 1 of 2)

| Keyword | Value |
|---|---|
| XSA_PASSWORD_LIFE | This data type specifies whether a new password that is being assigned to a user is temporary or permanent:<br><br>**PERMANENT**: Password is permanent and can be used for multiple logins.<br><br>**RESET**: Password is temporary and can be used only for a single login. |
| XSA_REVOKE_STATUS | The data type specifies an account status in the Managed System. The account status refers to a account owner ability to access the Managed System. A revoked account cannot log into the Managed System.<br><br>**REVOKED**: User is revoked and cannot log into the Managed System.<br><br>**ACTIVE**: User is not revoked and can log into the Managed System. |
| XSA_ADMIN_STATUS | This data type specifies the user’s administrative capacities in the Managed System.<br><br>A user with auditor capacity can review security definitions but cannot change them.<br><br>A user with administrator capacity can change security definitions. In some Managed Systems, the user can also review security definitions.<br><br>A user with all capacity can both review and change security definitions.<br><br>The exact scope of the administrative capacities associated with each of the attributes is Managed System-dependent.<br><br>**NONE**: User does not have the Managed System auditor or administrator attribute (that is, a regular user).<br><br>**AUDIT**: User has the Managed System auditor attribute.<br><br>**ADMIN**: User has the Managed System administrator attribute.<br><br>**ALL**: User has both Managed System auditor and administrator attributes. |

Table 35—AccountGet EntityHash Keyword Values Table (Sheet 2 of 2)

| Keyword | Value |
|---|---|
| XSA_DEF_GROUP_ACTION | This data type specifies the action to be performed on the old default group connection when the user's default group is updated. This data type is input to the Connector from IdentityIQ.<br><br>**DROP**: The connection between the user and the previous default group is deleted.<br><br>**KEEP**: The connection between the user and the previous default group is kept as a regular connection. |
| XSA_LOCK_STATUS | This data type specifies<br><br>**LOCKED**: TBD<br><br>**UNLOCKED**: TBD |

**Comments**: When using the return code XSA_RC_MORE, you can save the context information for consecutive calls to this function on the context Hash at XSA_ACTION_CONTEXT. For more information see “Maintaining the context” on page 70. Examples of the action context are the last account name sent, a pointer to the last place accessed in a file, and so on.

This function must include a call to XSA_WriteEntity.

## GroupGet

**Description**: GroupGet retrieves Group data from the Managed System.

Table 36—GroupGet Input Tables (Sheet 1 of 2)

| Table | Keyword | Description |
|---|---|---|
| XSA_OperationHash | XSA_ADMIN_ID | The administrator performing the action |
| | XSA_ADMIN_PASSWORD | The password of the administrator |
| | XSA_FILTER_TYPE | The type of filter (SINGLE, MANY or SUBTREE) |
| | XSA_GET_STAGE | The stage of the Get action (START, NEXT, or END) |
| | See Table 1— Common Keywords in the XSA_OperationHash table on page 18. | |
| XSA_ContextHash | XSA_CONNECTOR_CONTEXT | The connector level context |
| | XSA_ACTION_CONTEXT | The action level context or placeholder |
| | XSA_SESSION_CONTEXT | The session level context |
| | XSA_SYSTEM_CONTEXT | The Managed System level context |
| | XSA_TRANSACTION_CONTEXT | The transaction level context |

Table 36—GroupGet Input Tables (Sheet 2 of 2)

| Table | Keyword | Description |
|---|---|---|
| XSA_GetSearchHash | XSA_GROUP_NAME | Filter entities using this value |
| XSA_GetEntityHash | XSA_GROUP_NAME | Placeholder |
| | XSA_PARENT_GROUP | Placeholder |
| | User defined Keywords | |

Table 37—GroupGet Output Tables

| Table | Keyword | Description |
|---|---|---|
| XSA_GetEntityHash | XSA_GROUP_NAME | The name of the returned Group |
| | XSA_PARENT_GROUP | The name of the Parent Group of the returned Group |
| User defined Keywords | | |
| XSA_ContextHash | XSA_CONNECTOR_CONTEXT | The connector level context |
| | XSA_ACTION_CONTEXT | The action level context or placeholder |
| | XSA_SESSION_CONTEXT | The session level context |
| | XSA_SYSTEM_CONTEXT | The Managed System level context |
| | XSA_TRANSACTION_CONTEXT | The transaction level context |

Table 38—GroupGet Return Codes

| Code | Message |
|---|---|
| XSA_RC_OK (0) | Indicates that all entities were successfully retrieved |
| XSA_RC_MORE(1) | Indicates that this function should be called again to retrieve additional entities |
| XSA_RC_ERROR (8) | Indicates that an error occurred |

**Comments**: This function must include a call to XSA_WriteEntity.

## ConnectionGet

**Description**: ConnectionGet retrieves all Connections between Accounts and Groups on the Managed System.

Table 39—ConnectionGet Input Tables

| Table | Keyword | Description |
|---|---|---|
| XSA_OperationHash | XSA_ADMIN_ID | The administrator performing the action |
| | XSA_ADMIN_PASSWORD | The password of the administrator |
| | XSA_FILTER_TYPE | The type of filter (SINGLE, MANY, or PREFIX) |
| | XSA_GET_STAGE | The stage of the Get action (START, NEXT, or END) |
| | See Table 1— Common Keywords in the XSA_OperationHash table on page 18. | |
| XSA_ContextHash | XSA_CONNECTOR_CONTEXT | The connector level context |
| | XSA_ACTION_CONTEXT | The action level context or placeholder |
| | XSA_SESSION_CONTEXT | The session level context |
| | XSA_SYSTEM_CONTEXT | The Managed System level context |
| | XSA_TRANSACTION_CONTEXT | The transaction level context |
| XSA_GetSearchHash | XSA_ACCOUNT_NAME | Filter entities using this value |
| | XSA_GROUP_NAME | Filter entities using this value |
| XSA_GetEntityHash<br><br>See Table 42— Connection EntityHash Keyword Values Table on page 48 | XSA_ACCOUNT_NAME | Placeholder |
| | XSA_GROUP_NAME | Placeholder |
| | XSA_ADMIN_STATUS | Placeholder |
| | XSA_DEF_STATUS | Placeholder |
| | XSA_ACCOUNT_DEF_GROUP | Placeholder |
| | User defined Keywords | |

Table 40—ConnectionGet Output Tables

| Table | Keyword | Description |
|---|---|---|
| XSA_ContextHash | XSA_CONNECTOR_CONTEXT | The connector level context |
| | XSA_ACTION_CONTEXT | The action level context or placeholder |
| | XSA_SESSION_CONTEXT | The session level context |
| | XSA_SYSTEM_CONTEXT | The Managed System level context |
| | XSA_TRANSACTION_CONTEXT | The transaction level context |
| XSA_GetEntityHash<br><br>See Table 42—Connection EntityHash Keyword Values Table on page 48 | XSA_ACCOUNT_NAME | The Account name |
| | XSA_GROUP_NAME | The Group name |
| | XSA_ADMIN_STATUS | The Connection administrator status |
| | XSA_DEF_STATUS | The Connection default status |
| | XSA_ACCOUNT_DEF_GROUP | The Account default Group |
| | User-defined Keywords | |

Table 41—ConnectionGet Return Codes

| Code | Message |
|---|---|
| XSA_RC_OK (0) | Indicates that all entities were successfully retrieved |
| XSA_RC_MORE(1) | Indicates that this function should be called again to retrieve additional entities |
| XSA_RC_ERROR (8) | Indicates that an error occurred |

Table 42—Connection EntityHash Keyword Values Table

| Keyword | Value |
|---|---|
| XSA_ADMIN_STATUS | This data type specifies the administrative capacities of a user related to a group.<br><br>A user with auditor capacity can review security definitions related to the group but cannot change them.<br><br>A user with administrator capacity can change security definitions related to the group. In some Managed Systems, the user can also review security definitions.<br><br>A user with all capacity can both review and change security definitions related to the group.<br><br>The exact scope of security-related administrative capacities associated with each of the attributes is Managed System-dependent.<br><br>**NONE**: The user has no administrative capacity for the group.<br><br>**AUDIT**: The user is a security auditor of the group.<br><br>**ADMIN**: The user is a security administrator of the group.<br><br>**ALL**: The user is a security auditor and administrator of the group. |
| XSA_DEF_STATUS | This data type specifies whether or not a connection is the connection of a user to the user's default group.<br><br>**REGULAR**: This connection has no special attributes.<br><br>**DEFAULT_GROUP**: This connection is the connection to the default Group of this user. |

**Comments**: This function must include a call to XSA_WriteEntity.

## ContainerGet

**Description**: ContainerGet retrieves Container data from the Managed System.

Table 43—ContainerGet Input Tables (Sheet 1 of 2)

| Table | Keyword | Description |
|---|---|---|
| XSA_OperationHash | XSA_ADMIN_ID | The administrator performing the action |
| | XSA_ADMIN_PASSWORD | The password of the administrator |
| | XSA_FILTER_TYPE | The type of filter (SINGLE, MANY or SUBTREE) |
| | XSA_GET_STAGE | The stage of the Get action (START, NEXT, or END) |
| | See Table 1— Common Keywords in the XSA_OperationHash table on page 18. | |

Table 43—ContainerGet Input Tables (Sheet 2 of 2)

| Table | Keyword | Description |
|---|---|---|
| XSA_ContextHash | XSA_CONNECTOR_CONTEXT | The connector level context |
| | XSA_ACTION_CONTEXT | The action level context or placeholder |
| | XSA_SESSION_CONTEXT | The session level context |
| | XSA_SYSTEM_CONTEXT | The Managed System level context |
| | XSA_TRANSACTION_CONTEXT | The transaction level context |
| XSA_GetSearchHash | XSA_CONTAINER_NAME | Filter entities using this value |
| XSA_GetEntityHash | User defined Keywords | |

Table 44—ContainerGet Output Tables

| Table | Keyword | Description |
|---|---|---|
| XSA_ContextHash | XSA_CONNECTOR_CONTEXT | The connector level context |
| | XSA_ACTION_CONTEXT | The action level context or placeholder |
| | XSA_SESSION_CONTEXT | The session level context |
| | XSA_SYSTEM_CONTEXT | The Managed System level context |
| | XSA_TRANSACTION_CONTEXT | The transaction level context |
| XSA_GetEntityHash | User-defined Keywords | |

Table 45—ContainerGet Return Codes

| Code | Message |
|---|---|
| XSA_RC_OK (0) | Indicates that all entities were successfully retrieved |
| XSA_RC_MORE(1) | Indicates that this function should be called again to retrieve additional entities |
| XSA_RC_ERROR (8) | Indicates that an error occurred |

**Comments**: This function must include a call to XSA_WriteEntity.

## SystemPropertiesGet

**Description**: SystemPropertiesGet retrieves Managed System properties.

Table 46—SystemPropertiesGet Input Tables

| Table | Keyword | Description |
|---|---|---|
| XSA_OperationHash | XSA_ADMIN_ID | The administrator performing the action |
| | XSA_ADMIN_PASSWORD | The password of the administrator |
| | See Table 1— Common Keywords in the XSA_OperationHash table on page 18. | |
| XSA_ContextHash | XSA_CONNECTOR_CONTEXT | The connector level context |
| | XSA_SESSION_CONTEXT | The session level context |
| | XSA_SYSTEM_CONTEXT | The Managed System level context |
| | XSA_TRANSACTION_CONTEXT | The transaction level context |
| XSA_GetEntityHash | XSA_MIN_PASSWORD_LEN | Placeholder |
| | XSA_MAX_EXPIRE | Placeholder |
| | XSA_MAX_LOGINS | Placeholder |
| | User defined Keywords | |

Table 47—SystemPropertiesGet Output Tables

| Table | Keyword | Description |
|---|---|---|
| XSA_ContextHash | XSA_CONNECTOR_CONTEXT | The connector level context |
| | XSA_SESSION_CONTEXT | The session level context |
| | XSA_SYSTEM_CONTEXT | The Managed System level context |
| | XSA_TRANSACTION_CONTEXT | The transaction level context |
| XSA_GetEntityHash | XSA_MIN_PASSWORD_LEN | Minimum password length |
| | XSA_MAX_EXPIRE | Maximum number of days before expiration |
| | XSA_MAX_LOGINS | Maximum failed log-ins |
| | User-defined Keywords | |

Table 48—SystemPropertiesGet Return Codes

| Code | Message |
|---|---|
| XSA_RC_OK (0) | Indicates that system properties were successfully retrieved |
| XSA_RC_ERROR (8) | Indicates that an error occurred |

**Comments**: This function must include a call to XSA_WriteEntity.

## ResourceGet

**Description**: ResourceGet retrieves Resource data from the Managed System.

Table 49—ResourceGet Input Tables

| Table | Keyword | Description |
|---|---|---|
| XSA_OperationHash | XSA_ADMIN_ID | The administrator performing the action |
| | XSA_ADMIN_PASSWORD | The password of the administrator |
| | XSA_FILTER_TYPE | The type of filter (SINGLE, MANY or SUBTREE) |
| | XSA_GET_STAGE | The stage of the Get action (START, NEXT, or END) |
| | See Table 1— Common Keywords in the XSA_OperationHash table on page 18. | |
| XSA_ContextHash | XSA_CONNECTOR_CONTEXT | The connector level context |
| | XSA_ACTION_CONTEXT | The action level context or placeholder |
| | XSA_SESSION_CONTEXT | The session level context |
| | XSA_SYSTEM_CONTEXT | The Managed System level context |
| | XSA_TRANSACTION_CONTEXT | The transaction level context |

Table 50—ResourceGet Output Tables

| Table | Keyword | Description |
|---|---|---|
| XSA_GetEntityHash | User defined Keywords | |
| XSA_ContextHash | XSA_CONNECTOR_CONTEXT | The connector level context |
| | XSA_ACTION_CONTEXT | The action level context or placeholder |
| | XSA_SESSION_CONTEXT | The session level context |
| | XSA_SYSTEM_CONTEXT | The Managed System level context |
| | XSA_TRANSACTION_CONTEXT | The transaction level context |

Table 51—ResourceGet Return Codes

| Code | Message |
|---|---|
| XSA_RC_OK (0) | Indicates that all entities were successfully retrieved |
| XSA_RC_MORE(1) | Indicates that this function should be called again to retrieve additional entities |
| XSA_RC_ERROR (8) | Indicates that an error occurred |

**Comments**: This function must include a call to XSA_WriteEntity.

## ACEGet

**Description**: ACEGet retrieves Resource ACE data from the Managed System.

Table 52—ACEGet Input Tables

| Table | Keyword | Description |
|---|---|---|
| XSA_OperationHash | XSA_ADMIN_ID | The administrator performing the action |
| | XSA_ADMIN_PASSWORD | The password of the administrator |
| | XSA_FILTER_TYPE | The type of filter (SINGLE, MANY or SUBTREE) |
| | XSA_GET_STAGE | The stage of the Get action (START, NEXT, or END) |
| | See Table 1— Common Keywords in the XSA_OperationHash table on page 18. | |
| XSA_ContextHash | XSA_CONNECTOR_CONTEXT | The connector level context |
| | XSA_ACTION_CONTEXT | The action level context or placeholder |
| | XSA_SESSION_CONTEXT | The session level context |
| | XSA_SYSTEM_CONTEXT | The Managed System level context |
| | XSA_TRANSACTION_CONTEXT | The transaction level context |
| XSA_GetEntityHash | User defined Keywords | |

Table 53—ACEGet Output Tables

| Table | Keyword | Description |
|---|---|---|
| XSA_GetEntityHash | XSA_ACE_USER_NAME | This indicates whether the ACE entity belongs to Account Table |
| | XSA_ACE_UG_NAME | User Group |
| | XSA_ACE_OE_NAME | Container |
| | User defined Keywords | |
| XSA_ContextHash | XSA_CONNECTOR_CONTEXT | The connector level context |
| | XSA_ACTION_CONTEXT | The action level context or placeholder |
| | XSA_SESSION_CONTEXT | The session level context |
| | XSA_SYSTEM_CONTEXT | The Managed System level context |
| | XSA_TRANSACTION_CONTEXT | The transaction level context |

Table 54—ACEGet Return Codes

| Code | Message |
|---|---|
| XSA_RC_OK (0) | Indicates that all entities were successfully retrieved |
| XSA_RC_MORE(1) | Indicates that this function should be called again to retrieve additional entities |
| XSA_RC_ERROR (8) | Indicates that an error occurred |

**Comments**: This function must include a call to XSA_WriteEntity.

## Query Functions

This section describes the various query functions.

### Implementing Query Functions

Query functions are Connector functions that check certain Managed System data. The entire operation is performed in a single call to the function. A return code indicates the result of the query. Data is passed to the Query functions in tables like all the connector functions. These tables contain context and operation data, and the Information for the query it self resides on the XSA_GetSearchHash. For more Information on Operation data, see Table 1— Common Keywords in the XSA_OperationHash table on page 18.

### VerifyAdminPassword

**Description**: Verifies the password of an Administrator. This function is called when a Managed System Administrator is added or updated using the IdentityIQ – to verify the Administrator password.

This function is mandatory and must be implemented.

Table 55—VerifyAdminPassword Get Input Tables

| Table | Keyword | Description |
|---|---|---|
| XSA_OperationHash | See Table 1— Common Keywords in the XSA_OperationHash table on page 18. | |
| XSA_ContextHash | Dynamic | You can get here with different contexts, according to when the function was called |
| XSA_GetSearchHash | XSA_ACCOUNT_NAME | Filter entities using this value |
| | XSA_PASSWORD | Filter entities using this value |

Table 56—VerifyAdminPasswordGet Output Tables

| Table | Keyword | Description |
|---|---|---|
| XSA_ContextHash | XSA_SYSTEM_CONTEXT | The Managed System level context |

Table 57—VerifyAdminPasswordGet Return Codes

| Code | Message |
|---|---|
| XSA_RC_OK (0) | Indicates that the password was verified successfully |
| XSA_RC_ERROR (8) | Indicates that the password was incorrect or some other error |

### VerifyAccountPassword

**Description**: Verifies the password of an Account. This function call is initiated by Passport or Password manager for a user self registration and is used to verify the Account password on the Managed System.

Table 58—VerifyAccountPasswordGet Input Tables

| Table | Keyword | Description |
|---|---|---|
| XSA_OperationHash | See Table 1— Common Keywords in the XSA_OperationHash table on page 18. | |
| XSA_ContextHash | XSA_SYSTEM_CONTEXT | The Managed System level context |
| | XSA_CONNECTOR_CONTEXT | The Connector level context |
| | XSA_TRANSACTION_CONTEXT | The Transaction level context |
| | XSA_SESSION_CONTEXT | The Session level context |
| XSA_GetSearchHash | XSA_ACCOUNT_NAME | Filter entities using this value |
| | XSA_PASSWORD | Filter entities using this value |

Table 59—VerifyAccountPassworGet Output Tables

| Table | Keyword | Description |
|---|---|---|
| XSA_ContextHash | XSA_SYSTEM_CONTEXT | The Managed System level context |
| | XSA_CONNECTOR_CONTEXT | The Connector level context |
| | XSA_TRANSACTION_CONTEXT | The Transaction level context |
| | XSA_SESSION_CONTEXT | The Session level context |

Table 60—VerifyAccountPassworGet Return Codes

| Code | Message |
|---|---|
| XSA_RC_OK (0) | Indicates that the password was verified successfully |
| XSA_RC_ERROR (8) | Indicates that the password was incorrect or some other error |

## Set functions

This section describes the various Set functions.

### Implementing Set functions

Set functions are Connector functions that modify Managed System data. They can be used to insert new data, or to update or delete existing data in the Managed System.

Data is passed to the Set functions in tables. These tables contain context and operation data, as well as entity identification and Keyword data. For more information, see Table 1— Common Keywords in the XSA_OperationHash table on page 18. The entire operation may be performed in a single call to the function. A return code indicates the result of the operation.

New data is added by Add transactions. Add transactions receive the full details of the new entity to be added.

Existing data is modified by Update transactions. Update transactions receive only data that was changed for the entity.

Existing data is deleted by Delete transactions. Delete transactions receive no entity fields or data, other than entity identification.

### Set functions input tables

In addition to the input described in Table 1— Common Keywords in the XSA_OperationHash table on page 18, all Set functions (SystemPropertiesSet, AccountAdd, AccountUpdate, AccountDelete, GroupAdd, GroupUpdate, GroupDelete, ConnectionAdd , ConnectionUpdate, ConnectionDelete,ContainerAdd , ContainerUpdate, ContainerDelete, ResourceAdd, ResourceUpdate, ResourceDelete, ResourceGet,AceAdd, AceUpdate, AceDelete, AceGet) receive the following tables:

- XSA_SetIDHash: This table contains identification data for the entity currently being manipulated. Its fields are non-modifiable.
- XSA_SetKwdHash: This table contains all of the Keywords and values that are currently being modified.

### Set functions output tables

Set functions can modify only the XSA_ContextHash tables.

## Set functions reference

This section describes the various Ser functions references.

### AccountAdd

**Description**: AccountAdd creates a new Account on the Managed System. If the Account already exists on the Managed System, the function should return an error (XSA_RC_ERROR). If default Groups are supported in the Managed System, the function creates a connection between the new Account and the default Group.

Table 61—AccountAdd Input Tables (Sheet 1 of 2)

| Table | Keyword | Description |
|---|---|---|
| XSA_OperationHash | XSA_ADMIN_ID | The administrator performing the action |
| | XSA_ADMIN_PASSWORD | The password of the administrator |
| | See Table 1— Common Keywords in the XSA_OperationHash table on page 18. | |

Table 61—AccountAdd Input Tables (Sheet 2 of 2)

| Table | Keyword | Description |
|---|---|---|
| XSA_ContextHash | XSA_CONNECTOR_CONTEXT | The connector level context |
| | XSA_SESSION_CONTEXT | The session level context |
| | XSA_SYSTEM_CONTEXT | The Managed System level context |
| | XSA_TRANSACTION_CONTEXT | The transaction level context |
| XSA_SetIdHash | XSA_ACCOUNT_NAME | The Account name |
| XSA_SetKwdHash<br><br>See Table 35—AccountGet EntityHash Keyword Values Table on page 43 | XSA_PASSWORD | The Account password |
| | XSA_PASSWORD_LIFE | The Account password life |
| | XSA_REVOKE_STATUS | The Account status |
| | XSA_ADMIN_STATUS | The Account administrator status |
| | XSA_DEF_GROUP | The Account default Group name |
| | XSA_DEF_GROUP_ACTION | The Account default Group action |
| | User-defined Keywords | |

Table 62—AccountAdd Output Table

| Table | Keyword | Description |
|---|---|---|
| XSA_ContextHash | XSA_CONNECTOR_CONTEXT | The connector level context |
| | XSA_SESSION_CONTEXT | The session level context |
| | XSA_SYSTEM_CONTEXT | The Managed System level context |
| | XSA_TRANSACTION_CONTEXT | The transaction level context |

Table 63—AccountAdd Return Codes

| Code | Message |
|---|---|
| XSA_RC_OK (0) | Indicates that the Account was added successfully |
| XSA_RC_ERROR(8) | Indicates that an error occurred |

## AccountUpdate

**Description**: AccountUpdate updates the details of an existing Account on the Managed System. If the Account does not exist on the Managed System, the function should return an error (XSA_RC_ERROR).

If the default Group field for the Account is updated, the function connects the Account to the new default Group and sets it as the default connection. The connection of the Account to the previous default Group is deleted or kept as a regular connection according to the value XSA_DEF_GROUP_ACTION found on input table XSA_SetKwdHash.

Table 64—AccountUpdate Input Tables

| Table | Keyword | Description |
|---|---|---|
| XSA_OperationHash | XSA_ADMIN_ID | The administrator performing the action |
| | XSA_ADMIN_PASSWORD | The password of the administrator |
| | See Table 1— Common Keywords in the XSA_OperationHash table on page 18.s | |
| XSA_ContextHash | XSA_CONNECTOR_CONTEXT | The connector level context |
| | XSA_SESSION_CONTEXT | The session level context |
| | XSA_SYSTEM_CONTEXT | The Managed System level context |
| | XSA_TRANSACTION_CONTEXT | The transaction level context |
| XSA_SetIdHash | XSA_ACCOUNT_NAME | The Account name |
| XSA_SetKwdHash<br><br>See Table 35— AccountGet EntityHash Keyword Values Table on page 43 | XSA_PASSWORD | The Account password |
| | XSA_PASSWORD_LIFE | The Account password life |
| | XSA_REVOKE_STATUS | The Account status |
| | XSA_ADMIN_STATUS | The Account administrator status |
| | XSA_DEF_GROUP | The Account default Group name |
| | XSA_DEF_GROUP_ACTION | The Account default Group action |
| | User-defined Keywords | |

Table 65—AccountUpdate Output Table

| Table | Keyword | Description |
|---|---|---|
| XSA_ContextHash | XSA_CONNECTOR_CONTEXT | The connector level context |
| | XSA_SESSION_CONTEXT | The session level context |
| | XSA_SYSTEM_CONTEXT | The Managed System level context |
| | XSA_TRANSACTION_CONTEXT | The transaction level context |

Table 66—AccountUpdate Return Codes

| Code | Message |
|---|---|
| XSA_RC_OK (0) | Indicates that the Account was updated successfully |
| XSA_RC_ERROR(8) | Indicates that an error occurred |

## AccountDelete

**Description**: AccountDelete deletes an Account on the Managed System. If the Account is connected to any Groups, the function deletes all of this Account's Account to Group connections in the Managed System.

Table 67—AccountDelete Input Tables

| Table | Keyword | Description |
|---|---|---|
| XSA_OperationHash | XSA_ADMIN_ID | The administrator performing the action |
| | XSA_ADMIN_PASSWORD | The password of the administrator |
| | See Table 1— Common Keywords in the XSA_OperationHash table on page 18. | |
| XSA_ContextHash | XSA_CONNECTOR_CONTEXT | The connector level context |
| | XSA_SESSION_CONTEXT | The session level context |
| | XSA_SYSTEM_CONTEXT | The Managed System level context |
| | XSA_TRANSACTION_CONTEXT | The transaction level context |
| XSA_SetIdHash | XSA_ACCOUNT_NAME | The Account name |

Table 68—AccountDelete Output Tables

| Table | Keyword | Description |
|---|---|---|
| XSA_ContextHash | XSA_CONNECTOR_CONTEXT | The connector level context |
| | XSA_SESSION_CONTEXT | The session level context |
| | XSA_SYSTEM_CONTEXT | The Managed System level context |
| | XSA_TRANSACTION_CONTEXT | The transaction level context |

Table 69—AccountDelete Return Codes

| Code | Message |
|---|---|
| XSA_RC_OK (0) | Indicates that the Account was deleted successfully |
| XSA_RC_ERROR(8) | Indicates that an error occurred |

## ConnectionAdd

**Description**: ConnectionAdd creates a new Connection between an Account and a Group on the Managed System. If requested, the Account is also made an administrator or auditor (or both) of the Group. If the Connection already exists on the Managed System, the function should return an error (XSA_RC_ERROR).

Table 70—ConnectionAdd Input Tables

| Table | Keyword | Description |
|---|---|---|
| XSA_OperationHash | XSA_ADMIN_ID | The administrator performing the action |
| | XSA_ADMIN_PASSWORD | The password of the administrator |
| | See Table 1— Common Keywords in the XSA_OperationHash table on page 18.s | |
| XSA_ContextHash | XSA_CONNECTOR_CONTEXT | The connector level context |
| | XSA_SESSION_CONTEXT | The session level context |
| | XSA_SYSTEM_CONTEXT | The Managed System level context |
| | XSA_TRANSACTION_CONTEXT | The transaction level context |
| XSA_SetIdHash | XSA_ACCOUNT_NAME | The Account name |
| | XSA_GROUP_NAME | The Group name |
| XSA_SetKwdHash<br><br>See Table 42— Connection EntityHash Keyword Values Table on page 48 | XSA_ADMIN_STATUS | The Connection administrator status |
| | XSA_DEF_STATUS | The Connection default status |
| | XSA_ACCOUNT_DEF_GROUP | The Account default Group |
| | User-defined Keywords | |

Table 71—ConnectionAdd Output Tables

| Table | Keyword | Description |
|---|---|---|
| XSA_ContextHash | XSA_CONNECTOR_CONTEXT | The connector level context |
| | XSA_SESSION_CONTEXT | The session level context |
| | XSA_SYSTEM_CONTEXT | The Managed System level context |
| | XSA_TRANSACTION_CONTEXT | The transaction level context |

Table 72—ConnectionAdd Return Codes

| Code | Message |
|---|---|
| XSA_RC_OK (0) | Indicates that the Connection was added successfully |
| XSA_RC_ERROR(8) | Indicates that an error occurred |

## ConnectionUpdate

**Description**: ConnectionUpdate updates an existing Connection between an Account and a Group on the Managed System.

If requested, the Account is also made an administrator or auditor (or both) of the Group. If the Connection does not exist on the Managed System, the function should return an error (XSA_RC_ERROR).

Table 73—ConnectionUpdate Input Tables

| Table | Keyword | Description |
|---|---|---|
| XSA_OperationHash | XSA_ADMIN_ID | The administrator performing the action |
| | XSA_ADMIN_PASSWORD | The password of the administrator |
| | See Table 1— Common Keywords in the XSA_OperationHash table on page 18.s | |
| XSA_ContextHash | XSA_CONNECTOR_CONTEXT | The connector level context |
| | XSA_SESSION_CONTEXT | The session level context |
| | XSA_SYSTEM_CONTEXT | The Managed System level context |
| | XSA_TRANSACTION_CONTEXT | The transaction level context |
| XSA_SetIdHash | XSA_ACCOUNT_NAME | The Account name |
| | XSA_GROUP_NAME | The Group name |
| XSA_SetKwdHash | XSA_ADMIN_STATUS | The Connection administrator status |
| | XSA_DEF_STATUS | The Connection default status |
| | XSA_ACCOUNT_DEF_GROUP | The Account default Group |
| | User-defined Keywords | |

Table 74—ConnectionUpdate Output Table

| Table | Keyword | Description |
|---|---|---|
| XSA_ContextHash | XSA_CONNECTOR_CONTEXT | The connector level context |
| | XSA_SESSION_CONTEXT | The session level context |
| | XSA_SYSTEM_CONTEXT | The Managed System level context |
| | XSA_TRANSACTION_CONTEXT | The transaction level context |

Table 75—ConnectionUpdate Return Codes

| Code | Message |
|---|---|
| XSA_RC_OK (0) | Indicates that the Connection was updated successfully |
| XSA_RC_ERROR(8) | Indicates that an error occurred |

## ConnectionDelete

**Description**: ConnectionDelete deletes an existing Connection between an Account and a Group on the Managed System.

Table 76—ConnectionDelete Input Tables

| Table | Keyword | Description |
|---|---|---|
| XSA_OperationHash | XSA_ADMIN_ID | The administrator performing the action |
| | XSA_ADMIN_PASSWORD | The password of the administrator |
| | See Table 1— Common Keywords in the XSA_OperationHash table on page 18. | |
| XSA_ContextHash | XSA_CONNECTOR_CONTEXT | The connector level context |
| | XSA_SESSION_CONTEXT | The session level context |
| | XSA_SYSTEM_CONTEXT | The Managed System level context |
| | XSA_TRANSACTION_CONTEXT | The transaction level context |
| XSA_SetIdHash | XSA_ACCOUNT_NAME | The Account name |
| | XSA_GROUP_NAME | The Group name |

Table 77—ConnectionDelete Output Table

| Table | Keyword | Description |
|---|---|---|
| XSA_ContextHash | XSA_CONNECTOR_CONTEXT | The connector level context |
| | XSA_SESSION_CONTEXT | The session level context |
| | XSA_SYSTEM_CONTEXT | The Managed System level context |
| | XSA_TRANSACTION_CONTEXT | The transaction level context |

Table 78—ConnectionDelete Return Codes

| Code | Message |
|---|---|
| XSA_RC_OK (0) | Indicates that the Connection was deleted successfully |
| XSA_RC_ERROR(8) | Indicates that an error occurred |

## GroupAdd

**Description**: GroupAdd adds a Group on the Managed System. If the Group already exists on the Managed System, the function should return an error (XSA_RC_ERROR).

Table 79—GroupAdd Input Tables

| Table | Keyword | Description |
|---|---|---|
| XSA_OperationHash | XSA_ADMIN_ID | The administrator performing the action |
| | XSA_ADMIN_PASSWORD | The password of the administrator |
| | See Table 1— Common Keywords in the XSA_OperationHash table on page 18. | |
| XSA_ContextHash | XSA_CONNECTOR_CONTEXT | The connector level context |
| | XSA_SESSION_CONTEXT | The session level context |
| | XSA_SYSTEM_CONTEXT | The Managed System level context |
| | XSA_TRANSACTION_CONTEXT | The transaction level context |
| XSA_SetIdHash | XSA_GROUP_NAME | The Group name |
| XSA_SetKwdHash | XSA_PARENT_GROUP | The Parent Group name |
| | User-defined Keywords | |

Table 80—GroupAdd Output Table

| Table | Keyword | Description |
|---|---|---|
| XSA_ContextHash | XSA_CONNECTOR_CONTEXT | The connector level context |
| | XSA_SESSION_CONTEXT | The session level context |
| | XSA_SYSTEM_CONTEXT | The Managed System level context |
| | XSA_TRANSACTION_CONTEXT | The transaction level context |

Table 81—GroupAdd Return Codes

| Code | Message |
|---|---|
| XSA_RC_OK (0) | Indicates that the Group was added successfully |
| XSA_RC_ERROR(8) | Indicates that an error occurred |

## GroupUpdate

**Description**: GroupUpdate updates a Group on the Managed System. If the Group does not exist on the Managed System, the function should return an error (XSA_RC_ERROR).

Table 82—GroupUpdate Input Tables

| Table | Keyword | Description |
|---|---|---|
| XSA_OperationHash | XSA_ADMIN_ID | The administrator performing the action |
| | XSA_ADMIN_PASSWORD | The password of the administrator |
| | See Table 1— Common Keywords in the XSA_OperationHash table on page 18. | |
| XSA_ContextHash | XSA_CONNECTOR_CONTEXT | The connector level context |
| | XSA_SESSION_CONTEXT | The session level context |
| | XSA_SYSTEM_CONTEXT | The Managed System level context |
| | XSA_TRANSACTION_CONTEXT | The transaction level context |
| XSA_SetIdHash | XSA_GROUP_NAME | The Group name |
| XSA_SetKwdHash | XSA_PARENT_GROUP | The Parent Group name |
| | User-defined Keywords | |

Table 83—GroupUpdate Output Table

| Table | Keyword | Description |
|---|---|---|
| XSA_ContextHash | XSA_CONNECTOR_CONTEXT | The connector level context |
| | XSA_SESSION_CONTEXT | The session level context |
| | XSA_SYSTEM_CONTEXT | The Managed System level context |
| | XSA_TRANSACTION_CONTEXT | The transaction level context |

Table 84—GroupUpdate Return Codes

| Code | Message |
|---|---|
| XSA_RC_OK (0) | Indicates that the Group was updated successfully |
| XSA_RC_ERROR(8) | Indicates that an error occurred |

## GroupDelete

**Description**: GroupDelete deletes a Group on the Managed System.

Table 85—GroupDelete Input Tables (Sheet 1 of 2)

| Table | Keyword | Description |
|---|---|---|
| XSA_OperationHash | XSA_ADMIN_ID | The administrator performing the action |
| | XSA_ADMIN_PASSWORD | The password of the administrator |
| | See Table 1— Common Keywords in the XSA_OperationHash table on page 18. | |

Table 85—GroupDelete Input Tables (Sheet 2 of 2)

| Table | Keyword | Description |
|---|---|---|
| XSA_ContextHash | XSA_CONNECTOR_CONTEXT | The connector level context |
| | XSA_SESSION_CONTEXT | The session level context |
| | XSA_SYSTEM_CONTEXT | The Managed System level context |
| | XSA_TRANSACTION_CONTEXT | The transaction level context |
| XSA_SetIdHash | XSA_GROUP_NAME | The Group name |

Table 86—GroupDelete Output Tables

| Table | Keyword | Description |
|---|---|---|
| XSA_ContextHash | XSA_CONNECTOR_CONTEXT | The connector level context |
| | XSA_SESSION_CONTEXT | The session level context |
| | XSA_SYSTEM_CONTEXT | The Managed System level context |
| | XSA_TRANSACTION_CONTEXT | The transaction level context |

Table 87—GroupDelete Return Codes

| Code | Message |
|---|---|
| XSA_RC_OK (0) | Indicates that the Group was deleted successfully |
| XSA_RC_ERROR(8) | Indicates that an error occurred |

## ResourceAdd

**Description**: ResourceAdd adds a Resource on the Managed System. If the Resource already exists on the Managed System, the function should return an error (XSA_RC_ERROR).

Table 88—ResourceAdd Input Tables (Sheet 1 of 2)

| Table | Keyword | Description |
|---|---|---|
| XSA_OperationHash | XSA_ADMIN_ID | The administrator performing the action |
| | XSA_ADMIN_PASSWORD | The password of the administrator |
| | See Table 1— Common Keywords in the XSA_OperationHash table on page 18. | |
| XSA_ContextHash | XSA_CONNECTOR_CONTEXT | The connector level context |
| | XSA_SESSION_CONTEXT | The session level context |
| | XSA_SYSTEM_CONTEXT | The Managed System level context |
| | XSA_TRANSACTION_CONTEXT | The transaction level context |

Table 88—ResourceAdd Input Tables (Sheet 2 of 2)

| Table | Keyword | Description |
|---|---|---|
| XSA_SetKwdHash | User-defined Keywords<br><br>See Table 35— AccountGet EntityHash Keyword Values Table on page 43 | |

Table 89—ResourceAdd Output Table

| Table | Keyword | Description |
|---|---|---|
| XSA_ContextHash | XSA_CONNECTOR_CONTEXT | The connector level context |
| | XSA_SESSION_CONTEXT | The session level context |
| | XSA_SYSTEM_CONTEXT | The Managed System level context |
| | XSA_TRANSACTION_CONTEXT | The transaction level context |

Table 90—ResourceAdd Return Codes

| Code | Message |
|---|---|
| XSA_RC_OK (0) | Indicates that the Resource was added successfully |
| XSA_RC_ERROR(8) | Indicates that an error occurred |

## SystemPropertiesSet

**Description**: SystemPropertiesSet sets Managed System properties.

Table 91—SystemPropertiesSet Input Tables (Sheet 1 of 2)

| Table | Keyword | Description |
|---|---|---|
| XSA_OperationHash | XSA_ADMIN_ID | The administrator performing the action |
| | XSA_ADMIN_PASSWORD | The password of the administrator |
| | See Table 1— Common Keywords in the XSA_OperationHash table on page 18.s | |
| XSA_ContextHash | XSA_CONNECTOR_CONTEXT | The connector level context |
| | XSA_SESSION_CONTEXT | The session level context |
| | XSA_SYSTEM_CONTEXT | The Managed System level context |
| | XSA_TRANSACTION_CONTEXT | The transaction level context |

Table 91—SystemPropertiesSet Input Tables (Sheet 2 of 2)

| Table | Keyword | Description |
|---|---|---|
| XSA_SetKwdHash | XSA_MIN_PASSWORD_LEN | Minimum password length |
| | XSA_MAX_EXPIRE | Maximum days before expiration |
| | XSA_MAX_LOGINS | Maximum failed log-ins |
| | User-defined Keyword | |

Table 92—SystemPropertiesSet Output Table

| Table | Keyword | Description |
|---|---|---|
| XSA_ContextHash | XSA_CONNECTOR_CONTEXT | The connector level context |
| | XSA_SESSION_CONTEXT | The session level context |
| | XSA_SYSTEM_CONTEXT | The Managed System level context |
| | XSA_TRANSACTION_CONTEXT | The transaction level context |

Table 93—SystemPropertiesSet Return Codes

| Code | Message |
|---|---|
| XSA_RC_OK (0) | Indicates that the System Properties were updated successfully |
| XSA_RC_ERROR(8) | Indicates that an error occurred |

## Handling list keywords

The XSA_SetKwdHash and XSA_Get_EntityHash functions can contain list keywords, which will allow the manipulation of single dimension lists. The List handling is different for each supported language.

Table 94—List handling behavior

| Language | List definition | Behavior for an empty list |
|---|---|---|
| Java | Java vector | In Set functions an empty vector is received.<br><br>In Get functions an empty vector must be returned on the output table. |

# Chapter 5: Framework services

This chapter describes the various Framework services, such as printing messages and debugging the printout of messages, that are available to you.

This chapter presents the following topics:

## Overview

The Framework supplies the following services:

- Definitions
- Entity filling
- Entity synchronization
- Configuration parameter retrieval
- Messaging
- Debug display
- Context
- Unicode Support

### Names and constants

All names and constants (hash table names, return code values, and so forth) are defined by the Framework, in order to facilitate writing Connector code.

- **Java**
  - Constants are defined as class members in the XSA_Framework class.
  - Example: XSA_Framework.XSA_RC_OK.

### Entity synchronization

It is possible for the Connector or the Managed System to modify or add entities at the same time that they are acting on others. For example, you can configure your system so that whenever a new user is added to the

Managed System, the same user is also added to the POWER_USERS group. When this happens, IdentityIQ should be informed of the addition of the new Connection Entity. Use the XSA_WriteSynchronization service to send the new or modified entity to IdentityIQ.

## Entity retrieval

The Connector code should use a Get function to fill the values for one entity, and then call the Framework service XSA_WriteEntity (see "XSA_WriteEntity" on page 73). For more details about using Get functions, see "Get functions" on page 35.

## Controlling the code with parameters

The Connector code may require external (user-defined) Managed System configuration parameters, in order to enable various actions or to supply data, such as file names.

A Connector interfaces with each Managed System it manages by using a separate, distinct set of configuration data called a Managed System Configuration Set (MSCS). This is true whether the Connector manages one Managed System (requires one MSCS), or more than one Managed System (each MS requires one MSCS).

The actual MSCS is defined by the customer for a given Managed System using the IdentityIQ Configuration Console which uses the **MSCSTemplate.xml** file defined at step 5 of the Connector Factory.

The MSCS data created by the customer is not written to a single file. The SM will store the MSCS data for all the Connectors and Managed Systems that it manages in one aggregated file called the **MSCSparm** file.

To retrieve parameter values from this file, use the XSA_ReadParam service (see "XSA_ReadParam" on page 75).

## Testing and debugging the Connector

After the Connector code is written, it must be tested and debugged.

The testing should be done in phases:

- Phase 1 – Use the Connector Factory Test Step during the connector development to test the Connector.

  This is simple to perform, as the testing is integrated into the Connector Factory steps (see the Connector Factory User guide for more information). The Test step simulates in a relative way the transactions sent from IdentityIQ.
- Phase 2 – Test the Connector in a real world environment of IdentityIQ - generate and install the newly created Connector. Test it using real transactions from the IdentityIQ, and real offline interceptor.

The data displayed falls into the following categories:

- Errors discovered by the Framework

  You may have written code to perform invalid actions, such as trying to call the XSA_WriteEntity service during an update action.
- Exceptions caught by the Framework

  Connector code may terminate prematurely, due to interpretation errors or run-time errors, such as undefined variables, null pointers, and so forth. The Framework will catch the error and display all the information that is available from the language processor.
- Errors discovered by Connector components other than the Framework

  The Framework displays information regarding such errors (for example, an Account name that is invalid because it exceeds the maximum length).
- Informational messages

  The Framework provides information on the progress of processes, such as “download started”.
- Messages displayed by the Connector

  Messages written using the XSA_WriteMessage service.
- Debug information provided by the Connector

  Debug information written using the XSA_WriteDebug service.

This error and debugging data is displayed in the Connector Debug file.

## Connector debug file

This file contains errors and exceptions received from the Framework, along with error and debug information received from the Connector. (You can generate error and debug information in the Connector using the Framework services XSA_WriteDebugEnter, XSA_WriteDebugExit, XSA_WriteDebug and XSA_DebugHash.)

The debug information is controlled by the parameter XSA_DEBUG_LEVEL found in the **ConnectorParameters** configuration file.

The debug level is defined by the XSA_DEBUG_LEVEL parameter.

Framework errors and exceptions are written to the debug file, regardless of the debug level.

Debug information supplied via a call to the XSA_WriteDebug service is only written if the value defined for the level in the XSA_WriteDebug call (see “XSA_WriteDebug” on page 71) is equal to or less than the value defined for the XSA_DEBUG_LEVEL parameter (default = 0). This allows you to control the level of debugging.

**Note: XSA_DEBUG_LEVEL parameters cannot be changed for a specific Managed System; since they are at the Connector level, they can only be changed per Managed System Type.**

## Displaying the contents of a hash table for debugging

XSA_DebugHash is a service that displays the contents of a hash table. The contents are displayed in the Connector debug file. (See “XSA_DebugHash” on page 75.)

While writing the code:

1. Enter the XSA_DebugHash command into the code of the functions that you want to debug.
2. Generate and install the deployment connector.

For debugging the code:

1. Disable the platform in IdentityIQ (parameter changes are not detected when Connector is active).
2. Turn on debugging by changing the debug level in the configuration file (see "Controlling the code with parameters" on page 68).
3. Enable the platform in IdentityIQ.
4. Create a transaction in IdentityIQ.
5. Open the Connector debug file (see "Connector debug file" on page 69) to view the debug information that the Connector produced.
   Also view the messages and the alerts list in the IdentityIQ Transaction window.
6. Turn off debugging by setting the debug level to 0.

### Displaying errors and informational messages

If an internal error occurs, or if you would like to display lengthy messages or large amounts of technical data, use the XSA_WriteDebug service (see "XSA_WriteDebug" on page 71).

## Maintaining the context

The Connector code may be required to transfer data from the context of one action to another (for example, download Transaction you can save context relating to the different get Entity functions on the transaction Context), or to a different context within a continuing action, such as a Get action. The Framework enables this transfer of data by maintaining a context – some kind of object on the XSA_ContextHash table.

Actually the context mechanism for all contexts – except the session context - is like that of a simple Global Variable. The framework resets this variable when the time for it arrives – that is when after the appropriate term/end function is called (to allow the connector to terminate, clear, and close things), then Framework itself clears the context from the XSA_ContextHash table.

For the Session Context things are different since the entity Management functions can be called using the contexts of different Managed System Administrators, the Appropriate Session Contexts are saved per administrator, and are switched in the XSA_ContextHash table according to the right administrator before the function is called. And of course, like all contexts at the appropriate SessionTerm, the session context is deleted.

The Framework defines the following contexts:

- **Connector (Module) context**

  The value of this context is available during all actions. The hash table entry name is XSA_CONNECTOR_CONTEXT. This context can be modified at any time by the Connector code. It is maintained from the time that the ConnectorInit action begins until the completion of the ConnectorTerm action.

- **System Context**

  This is the Managed System Context. Its value is available during all actions performed on a particular Managed System. It is maintained from the call to ManageSystemInit to the call of ManageSystemTerm.

- **Session context**

  This is the administrator context and is maintained from the time that the SessionInit action begins until the completion of the SessionTerm action. It is available during all actions performed under the currently logged-in administrator.
- **Transaction Context**

  This is the Transaction context its value is available during all actions performed in the transaction It is maintained from the call to TransactionStart to the call of TransactionEnd.

  For example, AddAccount transaction will usually contain one call to AccountADD and one call to AccountGet between TransactionStart and TransactionEnd. During both calls the transaction context will be available.
- **Action context**

  The value of this context is only available during a single get action – from its start, and is deleted at its end. The hash table entry name is XSA_ACTION_CONTEXT.

## Unicode support

The IdentityIQ Connector Factory framework uses UTF-8 and translates the Java programming language encoding as follows:

- **Java**: All Java code uses UTF-16 by default.

# Framework services reference

**Note:** **When writing connector functions, you must always check the return code of the framework function it calls. If the status indicates an error, the Connector should also exit with an error**

## XSA_WriteDebug

**Description**: This service provides a display mechanism for debugging. As input it receives a level and a string. If the level is equal to or less than the value specified in the Connector configuration file (see "Controlling the code with parameters" on page 68), the string is written immediately to the debug file. The Framework defines several levels for your convenience, but any number may be used.

**Input**:

- Level number: The predefined values are:
  - XSA_DEBUG_ERROR(0) - Display always.
  - XSA_DEBUG_WARNING (10)
  - XSA_DEBUG_INFO( 20)
  - XSA_DEBUG_DETAIL(40)
- Text: The debug message string.

**Output**: The message string written to the debug file.

Table 95—XSA_Write Debug - Return codes

| Code | Message |
|---|---|
| XSA_RC_OK (0) | Indicates that the operation ended successfully |
| XSA_RC_ERROR(8) | Indicates that an error occurred |

Java example:

**// send a debug message After Connecting to a Database**

**XSA_Framework.XSA_WriteDebug(XSA_Framework.XSA_DEBUG_DETAIL, "Connecting to Db:  <" + dbName + "/" +”> result is: <” +res+”>”);**

## XSA_WriteDebugEnter

**Description**: This service provides a display mechanism for debugging when entering a function. As input it receives a level, and function name. If the level is equal to or less than the value specified in the Connector configuration file (see “Controlling the code with parameters” on page 68), the “Entering function …” diagnostic is written immediately to the debug file. The Framework defines several levels for your convenience, but any number may be used.

**Input:**

- Level number: The predefined values are:
    - XSA_DEBUG_ERROR(0) - Display always.
    - XSA_DEBUG_WARNING (10)
    - XSA_DEBUG_INFO( 20)
    - XSA_DEBUG_DETAIL(40)
- Function: The debugged function name

**Output**: The message string written to the debug file

Table 96—XSA_WriteDebugEnter - Return codes

| Code | Message |
|---|---|
| XSA_RC_OK (0) | Indicates that the operation ended successfully |
| XSA_RC_ERROR(8) | Indicates that an error occurred |

Java example:

**// send a debug message when entering the function**

**XSA_Framework.XSA_WriteDebugEnter(XSA_Framework.XSA_DEBUG_DETAIL,”AccountGet”);**

**Note: If you choose to use the XSA_WriteDebugEnter – you must use the appropriate XSA_WriteDebugExit when exiting the function. Failure to do this may create problems when using the Connector Factory Testing Facility at the Testing Step tree view.**

## XSA_WriteDebugExit

**Description**: This service provides a display mechanism for debugging when exiting a function. As input it receives a level, function name and return status. If the level is equal to or less than the value specified in the Connector configuration file (see "Controlling the code with parameters" on page 68), the "Exiting function …" diagnostic is written immediately to the debug file. The Framework defines several levels for your convenience, but any number may be used.

**Input**:

- Level number: The predefined values are:
  - XSA_DEBUG_ERROR(0) - Display always.
  - XSA_DEBUG_WARNING (10)
  - XSA_DEBUG_INFO( 20)
  - XSA_DEBUG_DETAIL(40)
- Function: The debugged function name.

  * rc – the function return status

**Output**: The message string written to the debug file.

Table 97—XSA_WriteDebugExit - Return codes

| Code | Message |
|---|---|
| XSA_RC_OK (0) | Indicates that the operation ended successfully |
| XSA_RC_ERROR(8) | Indicates that an error occurred |

Java example:

**// send a debug message when exiting the function**

**XSA_Framework.XSA_WriteDebugExit(XSA_Framework.XSA_DEBUG_DETAIL," AccountGet", rc);**

## XSA_WriteEntity

**Description**: Call this service from a Get action. The service reads data from the `XSA_GetEntityHash` table and adds the entity to an internal structure in Connector. Upon returning from the Get action, the entities are returned to IdentityIQ.

**Input**: None (The entity's data in the XSA_GetEntityHash table)

**Output**: N/A

Return code

Table 98—XSA_WriteEntity - Return codes

| Code | Message |
|---|---|
| XSA_RC_OK (0) | Indicates that the operation ended successfully |
| XSA_RC_ERROR(8) | Indicates that an error occurred |

Java example:

```
XSA_GetEntityHash.put("XSA_REVOKE_STATUS", "ACTIVE");

XSA_GetEntityHash.put("XSA_PASSWORD_LIFE", "PERMANENT");

rc = XSA_Framework.XSA_WriteEntity();
```

## XSA_WriteSynchronization

**Description**: This service receives data that identifies an entity and adds this data to an internal interceptions queue, indicating that IdentityIQ must be informed of a change to this entity. When the transaction finishes successfully, all of the interceptions are sent to IdentityIQ.

**Input**: The input is provided in Table 99—Input for XSA_WriteSynchronization.

Table 99—Input for XSA_WriteSynchronization

| Input | Description |
|---|---|
| Action | The type of action that occurred (ADD, UPDATE, or DELETE) |
| Entity_type | The entity type (ACCOUNT, GROUP, CONNECTION, CONTAINER or SYSTEM) |
| Entity_id | The Entity identifier:<br>•If  the Entity is an Account, use the Account identifier.<br>•If the Entity is a Group, use the Group identifier.<br>•If the Entity is a Connection, use the Account identifier and enter the Group identifier in Entity_id2.<br>•If the Entity is a System, supply either an empty string or a NULL pointer.<br>•If the Entity is a Container, use the Container identifier. |
| Entity_id2 | Used only for a Connection Entity<br>Specify the Group identifier.<br>Supply either an empty string or a NULL pointer for Account, Group, and System entities. |

Table 100—XSA_WriteSynchronization - Return codes

| Code | Message |
|---|---|
| XSA_RC_OK (0) | Indicates that the operation ended successfully |
| XSA_RC_ERROR(8) | Indicates that an error occurred |

Java example:

```
// To indicate that group called "MYGROUP" was changed -

XSA_Framework.XSA_WriteSynchronization("UPDATE", "GROUP",

  "MYGROUP");
```

## XSA_ReadParam

**Description**: Use this service to retrieve a parameter value from the Connector configuration files (see "Controlling the code with parameters" on page 68).

**Input**: Parameter name

**Output**: Parameter value. If no parameter is found, the following is returned:

- Java - Null

**Return value**: N/A

Java example: **TDB_Database.Init(XSA_Framework.XSA_ReadParam("database_dir"));**

## XSA_DebugHash

**Description**: Use this service to display the contents of a single hash table. The contents are written to the Connector debug file, and are subject to the debug level.

**Input**: Level: The debug level

- Description: This text appears as a heading for the contents.
- Hash table
    - Java - a reference to the hash table

**Output**: The contents of the hash table, displayed in the Connector debug file (see "Connector debug file" on page 69), provided that the input level is equal to or lower than the value for the Level parameter in the Connector configuration files (see "Controlling the code with parameters" on page 68).

**Return Value**: N/A

Java example:

```
// Print the contents of hash table
XSA_Framework.XSA_DebugHash(XSA_Framework.XSA_DEBUG_DETAIL,
  "XSA_GetEntityHash", XSA_GetEntityHash);
```

# Chapter 6: Java Programming language samples

This chapter provides samples and explains some issues specific to Java programming language.

**Predefined values**: Hash tables, Keyword names, and possible structure Keyword values are defined in the XSA_Framework class, located in XSA_FrameworkBase.java.

## Examples

### To return from the function

The following example indicates that the function ended successfully:

```
return XSA_Framework.XSA_RC_OK;
```

### To retrieve a value from a hash table

The following example determines whether the Get action returns a single Entity or more than one Entity, retrieves the value of the filter from the XSA_OperationHash table, and checks this value against a predefined value.

```
String filter = (String) IN_hashes[XSA_Framework.XSA_OPERATION_HASH].get(
XSA_Framework.XSA_OP_FILTER_STR);
if (filter.compareTo(XSA_Framework.XSA_OP_FILTER_SINGLE) != 0)
```

### To store a value in a hash table

The following example stores the current file position as the starting point for the next retrieval. The value is stored in the XSA_ContextHash table, under the ACTION context.

```
IN_hashes[XSA_Framework.XSA_CONTEXT_HASH].put(
XSA_Framework.XSA_CONTEXT_ACTION_STR,
Long.toString(record_location));
```

### To add an Entity to a reply

1. Use the following code to fill all the entries in the XSA_GET_ENTITY_HASH table.

```
for (Iterator i=IN_hashes[XSA_Framework.XSA_GET_ENTITY_HASH].keySet().iterator();
        i.hasNext(); )
    {
      field_name = (String)i.next();          // get key
      IN_hashes[XSA_Framework.XSA_GET_ENTITY_HASH].put(
                 field_name,
                 field_new_value);
     }
```

2. Send the Entity by calling:

```
if (XSA_Framework.XSA_WriteEntity() == XSA_Framework.XSA_RC_OK)
      ….;
    else
      ….;
```

### To write a message to a file and to the Transaction window

In the following example the Entity required for an update operation cannot be found:

**XSA_Framework.XSA_WriteMessage( "Entity to update not found.");**

### To write debug data

The following example writes debug data for a detailed analysis:
**XSA_Framework.XSA_WriteDebug(XSA_Framework.XSA_DEBUG_DETAIL,
"Searching for Entity " + id);**

### To create an Entity synchronization event

In the following example a Group is updated:

**XSA_Framework.XSA_WriteSynchronization("UPDATE", "GROUP", "group4");**

### To retrieve a parameter value from the Connector configuration file

The following example retrieves the value of a parameter named work_dir:

**Xwork_dir = XSA_Framework.XSA_ReadParam("work_dir");**

For more information about the configuration file, see "Controlling the code with parameters" on page 68.

### To display the contents of a hash table for debug purposes

The following example displays the contents of the XSA_OperationHash table:
**XSA_Framework.XSA_DebugHash(XSA_Framework.XSA_DEBUG_DETAIL,
"Show OP hash",
IN_hashes[XSA_Framework.XSA_OPERATION_HASH]);**

# Index