

CA Identity Manager

Programming Guide for Java

r12



This documentation and any related computer software help programs (hereinafter referred to as the "Documentation") is for the end user's informational purposes only and is subject to change or withdrawal by CA at any time.

This Documentation may not be copied, transferred, reproduced, disclosed, modified or duplicated, in whole or in part, without the prior written consent of CA. This Documentation is confidential and proprietary information of CA and protected by the copyright laws of the United States and international treaties.

Notwithstanding the foregoing, licensed users may print a reasonable number of copies of the Documentation for their own internal use, and may make one copy of the related software as reasonably required for back-up and disaster recovery purposes, provided that all CA copyright notices and legends are affixed to each reproduced copy. Only authorized employees, consultants, or agents of the user who are bound by the provisions of the license for the Product are permitted to have access to such copies.

The right to print copies of the Documentation and to make a copy of the related software is limited to the period during which the applicable license for the Product remains in full force and effect. Should the license terminate for any reason, it shall be the user's responsibility to certify in writing to CA that all copies and partial copies of the Documentation have been returned to CA or destroyed.

EXCEPT AS OTHERWISE STATED IN THE APPLICABLE LICENSE AGREEMENT, TO THE EXTENT PERMITTED BY APPLICABLE LAW, CA PROVIDES THIS DOCUMENTATION "AS IS" WITHOUT WARRANTY OF ANY KIND, INCLUDING WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NONINFRINGEMENT. IN NO EVENT WILL CA BE LIABLE TO THE END USER OR ANY THIRD PARTY FOR ANY LOSS OR DAMAGE, DIRECT OR INDIRECT, FROM THE USE OF THIS DOCUMENTATION, INCLUDING WITHOUT LIMITATION, LOST PROFITS, BUSINESS INTERRUPTION, GOODWILL, OR LOST DATA, EVEN IF CA IS EXPRESSLY ADVISED OF SUCH LOSS OR DAMAGE.

The use of any product referenced in the Documentation is governed by the end user's applicable license agreement.

The manufacturer of this Documentation is CA.

Provided with "Restricted Rights." Use, duplication or disclosure by the United States Government is subject to the restrictions set forth in FAR Sections 12.212, 52.227-14, and 52.227-19(c)(1) - (2) and DFARS Section 252.227-7014(b)(3), as applicable, or their successors.

All trademarks, trade names, service marks, and logos referenced herein belong to their respective companies.

Copyright © 2008 CA. All rights reserved.

CA Product References

This document references the following CA products:

- CA Identity Manager
- CA SiteMinder® Web Access Manager
- CA Security Command Center (SCC)
- CA Audit
- eTrust® Directory, also known as CA Directory

Contact Technical Support

For online technical assistance and a complete list of locations, primary service hours, and telephone numbers, contact Technical Support at <http://ca.com/support>.

Contents

Chapter 1: Identity Manager APIs	11
About the Documentation	11
API Installation	11
API Overview	12
Identity Manager APIs	12
Types of Custom Operations	13
Task Phases	15
Synchronous Phase	15
Synchronous Phase Operations	16
Asynchronous Phase	19
Asynchronous Phase Operations	20
Task Execution Summary	23
API Functional Model	25
Synchronous Phase Model	25
Asynchronous Phase Model	26
Model Components	27
 Chapter 2: Logical Attribute API	 31
Logical Attributes and Physical Attributes	31
Attribute Data Flow	32
Logical Representation of Physical Attributes	33
Logical Attribute Elements	34
Predefined Logical Attributes	35
Logical Attribute API Overview	38
Logical Attribute API Summary	38
Use Case Examples	39
API Components	40
Logical Attribute Handler API Model	41
Calling Sequence	42
Custom Messages	43
Sample Logical Attribute Handlers	43
Logical Attribute Configuration	44
Option Lists	45
Populating Option Lists	45
Logical Attribute Handlers for Option Lists	45
Adding an Option List to a Task Screen	46
Option List Configuration Summary	47

Option Lists at Run Time	47
Self-Registration Logical Attribute Handler	48
How a Self-Registration Handler Works	48
Writing the Self-Registration Handler	49
Self-Registration and the API Functional Model	50
Self-Registration Handler Configuration	50
Sample Self-Registration Handler	51
Logical Attribute Handler for a Forgotten Password	51
Writing a Custom Forgotten Password Handler	52
Forgotten Password Handler Configuration	53
Logical Attribute Implementation	53
Logical Attribute Handler at Run Time	54
Example: Assigning a Logical Attribute to a Field	54
Use Case Example: Formatting a Date Attribute	55
Screen-Defined Logical Attributes	58
Display the Configure Standard Profile Screen	59
Initializing Screen-Defined Logical Attributes	60
Updating Screen-Defined Logical Attributes	60

Chapter 3: Business Logic Task Handler API 63

API Overview	63
API Summary	63
Task Session	64
Scope of Business Logic Task Handlers	66
Use Case Examples	66
API Components	67
Business Logic Task Handler API Model	69
Sample Business Logic Task Handlers	69
Access to Managed Objects	70
Managed Object Process Diagram	71
Calling Sequence	72
Example: Calling a Business Logic Task Handler	73
Custom Messages	73
Informational Messages	73
Exception Messages	74
Custom Message Example	74
JavaScript Handler Implementation	74
JavaScript Method Reference	75
IMSAPI Class References	76
Configuring Business Logic Task Handlers	77
Global Business Logic Task Handlers	77
Task-Specific Business Logic Task Handlers	78

Chapter 4: Event Listener API **79**

Tasks and Events	79
Event Listener Overview	80
Event Listener Operations	81
Use Case Examples	82
API Components	83
Event Listener API Model	84
Event Listener Execution	84
Event Listener States	85
Event Listener State Diagram	86
Sample Event Listener	87
Order of Execution	87
Access to Managed Objects	88
Event Generation	89
Event Listener Configuration	89

Chapter 5: Workflow API **91**

Workflow API Overview	91
Workflow API Summary	92
Use Case Examples	92
API Components	94
Integration with Third-Party Applications	96
Participant Resolver API	97
About Participant Resolvers	97
API Components	99
Calling Sequence	101
Participant Resolver Configuration	102

Chapter 6: Notification Rule API **103**

API Overview	103
Notification Rule API Summary	103
Use Case Examples	104
Email Templates	104
API Components	105
Notification Rule API Model	106
Sample Notification Rule	106
Notification Rule Configuration	107

Chapter 7: Managed Objects **109**

Managed Object Overview	109
-------------------------------	-----

Managed Object Interfaces	110
Super Interfaces	111
Extended Interfaces.....	112
Managed Object Attributes	113
Retrieval Example	114
Access to Managed Object Data	114
Access to Objects in the Data Store.....	117
Providers	117
Retrieving Providers.....	118
Attribute and Permission Requests	119
Access to Attribute Data	119
Calling Sequence	120
Access to Objects in a Task	121
Parallels with Task Screen Operations	121
Retrieval of Managed Objects	122
Calling Sequence	123
Attribute Validation Type	123
Access to Event Objects.....	124
Events and Event Objects	124
Event Object Tables	128

Chapter 8: Support Objects 133

Identity Policies	133
IMEvent.....	134
IMEventName	134
IMSEException	134
Localizing Exception Messages	135
Localizer Class	136
Use the Localizer Class	137
PasswordCondition.....	137
Permissions Objects	137
Policies and Policy Conditions	138
PropertyDict.....	138
Provisioning Roles	139
ResultsContainer	139
RoleDisplayData	140
Search Objects	140
Search Filters and Constraints	140
Task.FieldId	141
TSContext	141
Type Objects	141
Validation Objects	142

Attribute Validation	143
Chapter 9: Compiling and Deploying	145
Compiling Source Files	145
Deploying Run Time Files	146
Chapter 10: Task Execution Web Service	147
What Are Web Services?	147
Web Service Protocols	147
What Is TEWS?	148
Remote Task Requests	149
How a Remote Request Is Processed	150
Task Operation Types	150
How Task Operations Work	151
A Task Operations Example	152
Web Service Configuration	152
Web Service Properties Screen	153
Web Services Properties	153
Enable Web Services for Individual Tasks	154
Client Application Development	155
WSDL Generation	156
SOAP Messages	157
Exception Messages	158
Authentication	158
Authorization	159
Administrator Specification	160
Administrator Parameters	160
The Channel Connection	162
Localization	162
Chapter 11: TEWS Sample Client Code	165
Web Service Development Workstation	165
Third-Party Software	165
Web Service Samples	166
Java Samples for Axis	167
Java Sample Classes	167
Build Script for Axis	168
Running the Java Classes	168

Appendix A: TEWS Client Migration	169
Migrating to Identity Manager r8.1	169
Namespace and Package Name Changes	169
Attribute Name Changes	169
Result and Status Proxy Values	169
Public and Protected Context Tasks	170
Locators and Ports	171
Migrating to Identity Minder r6.0	171
Disable Prior Versions	172
 Appendix B: TEWS Error Codes	 173
Task Session Error Codes	174
 Index	 175

Chapter 1: Identity Manager APIs

This section contains the following topics:

[About the Documentation](#) (see page 11)
[API Installation](#) (see page 11)
[API Overview](#) (see page 12)
[Identity Manager APIs](#) (see page 12)
[Types of Custom Operations](#) (see page 13)
[Task Phases](#) (see page 15)
[Task Execution Summary](#) (see page 23)
[API Functional Model](#) (see page 25)

About the Documentation

The integrated HTML version of the *CA Identity Manager Programming Reference* includes two parts:

- **Javadoc Reference**--Accessible through the Javadoc tab in the navigation pane. This tab applies to the Javadoc Reference only.
- **Programming Guide**--Accessible through the Content, Index, and Search tabs in the navigation pane. These tabs apply to the Programming Guide only.

The integrated *CA Identity Manager Programming Reference* also includes hyperlinks between the Javadoc Reference and the Programming Guide where necessary to cross-reference relevant information.

Note: The PDF version of the *CA Identity Manager Programming Guide* does not include the Javadoc Reference information.

API Installation

You install the Identity Manager APIs by installing the Identity Manager Administrative Tools, described in the *Identity Manager Installation Guide*.

The Administrative Tools include configuration files, scripts, utilities, and jar files that you use to compile custom objects with Identity Manager APIs and API samples. Administrative Tools are placed in the following default locations:

Windows: C:\Program Files\CA\IAM Suite\Identity Manager\tools

UNIX: HOME/CA/IAM_Suite/Identity_Manager/tools

API Overview

You can use the Identity Manager APIs to customize these features:

- The schema, which describes a user store's structure to Identity Manager.
- The "look and feel" of the user interface.
- User entry screens, which determine the fields and layout of each task screen.
- Validation of user data entry, through regular expression, JavaScript, or Java implementations.
- Workflow, which defines automated workflow processes. Create or modify processes by linking approvers and actions in the WorkPoint Process Designer.
- Email messages, which inform users of a task's status
- Task submission, which can be sent by a third-party application to the Identity Manager Task Execution Web Service (TEWS). TEWS processes the remote task request. Remote task requests comply with WSDL standards.

Identity Manager APIs

You can use the following APIs to customize Identity Manager:

Logical Attribute API

Displays an attribute differently than how it is stored physically in a user directory.

Use this API to work with the values of managed object attributes (such as the value of the employee_number or email attribute of a User object). For example:

- When a task screen is about to be displayed, convert an attribute value from its stored physical form to its displayed form.
- Automatically populate the value of an attribute, possibly through an external system.
- Validate user-supplied data on a task screen.
- After a task is submitted for execution, convert an attribute value from its displayed form on the task screen to its stored physical form.

Business Logic Task Handler API

Performs custom business logic at various times before the task is submitted, using any of the data associated with the submitted task or in the data store.

Workflow API

Provides the workflow engine with information about the Identity Manager event being executed. Using this information, a workflow script can make decisions about whether an approval is required for an event and if so, who should approve it. The workflow API also allows a workflow script to access and modify managed object attribute values.

Participant Resolver API

Specifies the list of participants who are authorized to approve or reject a workflow activity.

Event Listener API

Creates a custom event listener that listens for a specific Identity Manager event or group of events. When the event occurs, the event listener performs custom business logic during particular states of execution (Pre-approval, Approved, Rejected, Post-execution). Also generate new events within the current task.

Notification Rule API

Determines the users who should receive an email notification.

Email Template API

Includes event-specific information in an automated email notification.

Note: For more information about the Email Template API, see the *CA Identity Manager Administration Guide*.

For technical reference information about all Identity Manager APIs, see the Javadoc Reference.

Types of Custom Operations

This table lists the kinds of operations you can perform with Identity Manager APIs:

Custom Operation	Applicable API
Format data from a data source	Logical Attribute
Validate data entry	Logical Attribute
Add a drop-down list to a task screen	Logical Attribute

Check whether a group membership limit is reached	Business Logic Task Handler
Verify the job title of a new administrator	Business Logic Task Handler
Check for duplicate objects	Business Logic Task Handler
Disable a user account after the maximum failed login attempts	Business Logic Task Handler
Assign a user to a group during self-registration	Business Logic Task Handler
Assign “manager” attribute for a new user	Business Logic Task Handler
Determine workflow direction based on a User object attribute	Workflow
Update an Identity Manager object during a workflow process	Workflow
Validate an object from a workflow process using third-party data	Workflow
Assign a group’s administrators as participants	Participant Resolver
Add a user to Microsoft Exchange (specific event listener)	Event Listener
Create a rule for group assignments (specific event listener)	Event Listener
Update an RDBMS with user data (class-level event listener)	Event Listener
Post events to a JMS queue (all event listener)	Event Listener
Notify a single user	Notification Rule
Notify a group	Notification Rule

Task Phases

Identity Manager users are administrators who are assigned to roles. Each role contains tasks (such as Create User or Create Group) that users in that role can perform.

Identity Manager users can perform administrative operations on the following kinds of tasks:

- *Admin tasks* manage Identity Manager objects such as users, groups, organizations, roles, and other tasks.
- *Access tasks* are business functions performed in business application outside of Identity Manager.

Note: Identity Manager provides administrative control of access tasks, but it does not execute these tasks.

Admin task execution has a two phases:

- In the *synchronous phase*, the task has not yet been broken down into its component events.
- In the *asynchronous phase*, the task has been broken down into its component events.

Synchronous Phase

The synchronous phase performs data validation and manipulation.

It begins when the user selects a task to perform, or when an anonymous user requests self-registration or a password change.

It ends when data validation and manipulation operations are complete, and the task is sent to Identity Manager for processing.

Description

When a user logs in to Identity Manager, Identity Manager displays the tasks that the user can perform. A task is performed within a task session. A task session contains information such as the time that task execution began, the name and unique identifier of the administrator who is executing the task, and the organization where the task is being executed.

A task session also contains the Identity Manager managed object that is affected by the task. For example, a task session for a Create User task contains a User object consisting of profile information about the new user. In addition, the task session contains lists of any resources (such as roles and groups) that the task's managed object is associated with.

When the user selects a task to perform, the synchronous phase of task processing begins. The associated task screen is displayed, allowing the user to supply the data required for the task. In the synchronous phase, validation and manipulation of user-supplied data are performed. If validation errors occur, the user receives immediate feedback. The user can then correct the errors and resubmit the task.

More information:

[Managed Objects](#) (see page 109)

Synchronous Phase Operations

In the synchronous phase, the following operations are performed in the following order:

1. The User Selects a Task to Perform

After the user selects a task for processing, but before the user is presented with a list of possible subjects for the task, the following processing can occur:

Business processing through a business logic task handler's `handleStart()` method.

API: Business Logic Task Handler API.

2. The Task's Subject is Selected but Not Yet Displayed

After the user selects the subject of the task, but before the subject's data is displayed on the task screen, the following processing can occur:

- Business processing through a business logic task handler's `handleSetSubject()` method.

API: Business Logic Task Handler API.

The subject of a task is the object that is being directly affected by the task—for example, a User object in a Create User task.

- Default values for fields can be initialized with JavaScript.

During task screen configuration, default values can be hard-coded, or JavaScript can be provided to initialize fields dynamically.

- Fields associated with physical attributes are automatically populated with the corresponding data from the data store. Identity Manager performs this initialization.

3. Identity Manager Displays and Populates the Task Screen

After the user has selected the task to perform and the task's subject, Identity Manager displays the appropriate task screen containing the subject's data.

A task screen may contain fields associated with physical attributes or logical attributes:

Fields associated with logical attributes are populated with data provided by logical attribute handlers. For example, a department number may be converted from a numeric code (physical attribute) to its corresponding user-friendly name (logical attribute) before being displayed on the task screen.

API: Logical Attribute API.

4. Identity Manager Invokes Validation Rules During Data Entry

Identity Manager may execute validation rules during data entry. Validation rules are specified in the `directory.xml` file for an entire environment, or in a task screen for a particular task. Validation rules are not performed by logical attribute handlers.

During data entry, validation rules can be invoked in any of these ways:

- A tab's Validate button is clicked.
- The user changes tabs on the task screen.
- A value is modified in a field for which Validate on change is enabled.

5. User Submits the Task Screen

After the user finishes supplying data on the task screen and clicks Submit, Identity Manager performs the following operations in the following order:

- Invokes validation rules.
Validation rules can be implemented as regular expressions, JavaScript, or Java classes.
- Calls `validate()` in each logical attribute handler. This method performs field-level validation.

Field-level validation takes place within the scope of the user-supplied attribute value only. For example, if you are validating a date, you can validate the date itself, but you cannot compare the date against other information in the task. Validating data against other data in the task is called task-level validation.

API: Logical Attribute API.

- Calls `toPhysical()` in each logical attribute handler. This method converts logical attributes back to their corresponding physical attributes.

API: Logical Attribute API.

- Calls the `handleValidation()` method in business logic task handlers to perform task-level validation.

Task-level validation lets you validate field data against any information in the task session or in the data store. For example, during a Create User task, Identity Manager calls a predefined business logic task handler that verifies whether the user ID for the new user is unique.

Business logic task handlers can be called at the following times during the synchronous phase of task processing:

- When a task session is created (to initialize or process task data)
- After selecting a subject from the search result, but before presenting data to a user
- After a user submits a task, but before any security checks
- After security checks are done, but before the asynchronous phase

API: Business Logic Task Handler API.

6. Identity Manager Displays Validation Results

If validation errors occur, the errors are reported to the user immediately. The user is allowed to correct the errors and resubmit the task.

7. Identity Manager Instantiates Events for the Task

After all validation is performed successfully, Identity Manager breaks down the task into the individual transactions that occur during the execution of the task.

Each transaction is represented by an *event*. For example, a Create User task may consist of the following events:

- `CreateUserEvent`. The event representing the creation of the User object.
- `AddToGroupEvent`. If the newly created user is being added to a group, this event represents the addition.

Event processing allows a finer degree of control over the execution of a task. For example, workflow approvals can be required for individual events, so that the rejection of one event may not impact the execution of other events or the overall task.

- After events are instantiated for the task, the following processing can occur:
- Business processing through a business logic task handler's `handleSubmission()` method.

API: Business Logic Task Handler API.

After Identity Manager calls this method for all associated business logic task handlers, the synchronous phase is complete. The task enters the asynchronous phase for processing.

Note: Validation rules can be implemented as regular expressions, JavaScript, or Java classes. For more information, see the *CA Identity Manager Configuration Guide*.

Asynchronous Phase

The asynchronous phase perform the task by executing the events associated with the task. Optionally, perform workflow approvals and send email notifications.

It begins when data validation and manipulation performed in the synchronous phase are complete.

It ends when task processing is complete.

Description

When the data validation and manipulation performed in the synchronous phase are complete, and the task's events have been instantiated, the asynchronous phase, the task processing, begins.

Both logical attribute values and physical attribute values are available to custom objects for processing.

Feedback resulting from an error may not occur immediately, as it does in the synchronous phase.

Asynchronous Phase Operations

In the asynchronous phase, the events that were instantiated for the task during the synchronous phase are posted and cannot be recalled.

Identity Manager completes the task after all associated events are completed.

All operations are optional except the execution of the event.

The following operations are performed for each event, in the following order:

1. Identity Manager Posts the Events

At the beginning of the asynchronous phase of task execution, Identity Manager posts the events that were instantiated for the task during the synchronous phase. At this point, the events are committed for the task and cannot be recalled.

If workflow is enabled, and one or more of the task's events are mapped to a workflow process, the task comes under workflow control. The task remains in a pending state until the mapped workflow processes are completed.

2. Event Listeners Perform Pre-Approval Attribute Updates

The Pre-approval state occurs before a workflow process begins, or before the automatic approval of an event that is not associated with a workflow process. During the Pre-approval state, a custom event listener can update attributes in Identity Manager managed objects associated with the event.

API: Event Listener API.

3. Custom Objects Perform Operations During Workflow Processing

If workflow is enabled for the Identity Manager environment, and an event is mapped to a workflow process, the activities in the workflow process must be completed before the event (and ultimately, the associated task) can be completed.

Note: Custom objects cannot update attributes in Identity Manager objects during workflow processing.

During workflow operations, the following custom processing can occur:

- Specify the participants (approvers) for a workflow activity.

API: Participant Resolver API.

- Retrieve information about the event and use the information to determine the path of the workflow process.

For example, during a Create User task, a custom object could retrieve the user's title. If the title is Manager, a two-step approval is required (from HR and from a corporate officer). If the user's title is other than Manager, a one-step approval is required, just from HR.

API: Workflow API.

- Retrieve information from a third-party application and use the information to determine the path of the workflow process.

This type of custom processing, called asynchronous validation, lets you add automated activities to a workflow process. The automated activity calls the third-party application and requests the information. The third-party application then returns the information to the activity, where the information is evaluated to determine the course of subsequent workflow processing.

Note: A workflow process can have an unlimited number of manual and automated activities. For information about workflow activities, see the *CA Identity Manager Operations Guide*.

4. Event Listeners Perform Pre-Execution Attribute Updates

A workflow process results in either the approval or rejection of the associated event, depending upon the outcome of the individual activities in the workflow process. When a workflow process is approved, Identity Manager executes the associated event.

After a workflow process is approved or rejected, but before Identity Manager executes an approved event, a custom object can modify the attributes of the managed objects in the event. However, if the workflow process is rejected, the event is never executed, and any modifications to the managed object in the event are not written to the data store.

Pre-execution attribute updates can also be made for events that are not under workflow control. An event that is not under workflow control is automatically approved.

API: Event Listener API.

5. Identity Manager Executes the Event

Event execution is complete when all operations associated with the event have been completed.

If managed object data associated with an event needs to be written to the data store, it is written after execution of the event is complete.

No Identity Manager APIs are applicable to this operation.

6. Event Listeners Perform Post-Execution Operations

After the execution of an event is complete, an event listener can perform operations such as the following:

- Performing rule-based role or group provisioning. For example, suppose users who are assigned to the group QA should automatically be assigned to the group Engineering. An event listener that is "listening for" the event `AddToGroupEvent` in the Post-event state can:
 - Check whether the group that the user has been added to is QA.
 - If the user was added to QA, add the user to Engineering.
 - Generate another `AddToGroupEvent` for the addition of the user to Engineering. This allows the additional `AddToGroupEvent` to be subject to its own workflow and auditing operations.
- Notifying third-party applications that the event has executed. Notification that an event has executed may be performed for auditing purposes or to allow the third-party application to perform any post-execution processing that it may require.

API: Event Listener API.

7. Identity Manager Generates Email Notifications

Email notifications report the status of tasks and events.

Email notifications are generated from templates. A template can generate event-specific messages through JavaScript calls to the Email Template API. Further, you can write a custom object that determines the list of recipients for email notifications.

An email's recipients are determined by the template used to generate the email. For example:

- Emails based on the Pending template are sent to all users who are potential participants in the associated workflow activity.
- All templates can call the `getNotifiers()` method to determine the list of recipients. This method passes to Identity Manager the name of a Java object called a notification rule that determines the list. The method can pass the name of a predefined notification rule (`ADMIN`, `USER`, or `USER_MANAGER`) or a custom notification rule.

APIs: Email Template API, Notification Rule API.

Task Execution Summary

This section summarizes the sequence of operations that can occur when an administrator executes a Modify User task, including adding the end user to a group. In this example, workflow and email notifications are enabled.

1. The administrator selects Modify User from the list of tasks that he is authorized to perform.
2. Identity Manager creates a task session.

API: Identity Manager calls `handleStart()` in applicable business logic task handlers (those that are defined in the environment or are associated with the task).

3. The administrator selects a particular User object to modify.

API: Identity Manager calls `handleSetSubject()` in applicable business logic task handlers.

4. Identity Manager retrieves the User object.

API: If any physical attributes in the User object need to be converted to logical attributes, logical attribute handlers perform the conversions.

5. Identity Manager displays the user data on a task screen.

6. The administrator modifies some of the user data on the task screen, and also adds the user to a group. The administrator then clicks Submit.

API: Identity Manager executes validation rules for all applicable fields, and then calls logical attribute handlers to validate logical attribute fields. If a validation fails, an exception is thrown, an error is displayed, and the administrator must resubmit the task.

API: Logical attribute handlers convert logical attribute values back to physical attribute values.

API: Identity Manager calls `handleValidation()` in business logic task handlers to perform task-level validation and any other business logic that may be required. If any handler processing fails, an exception is thrown, an error is displayed, and the administrator must resubmit the task.

7. Identity Manager instantiates the events `ModifyUserEvent` and `AddToGroupEvent`. In this example, both of these events are mapped to workflow processes.

API: Identity Manager calls `handleSubmission()` in applicable business logic task handlers.

After this method is called in all business logic task handlers defined in the environment or associated with the task, the task enters the asynchronous phase for processing.

8. Identity Manager posts the events instantiated in the synchronous phase. Events can no longer be recalled.

9. Identity Manager begins processing the event `ModifyUserEvent`.
API: Identity Manager calls the `before()` method in each event listener for the event `ModifyUserEvent`. This method is called even if the event is not mapped to a workflow process.
10. The workflow process begins for the event `ModifyUserEvent`.
API: The Participant Resolver API determines the participants (approvers) for the workflow activity.
API: The Workflow API can be called to pass information about the event or information from a third-party application back to the workflow process. A workflow process activity evaluates the information to determine the next activity to be performed.
11. Identity Manager sends email notification about the pending event.
API: An email notification can include event-specific information through JavaScript calls to the Email Template API. Also, email recipients can include users who are specified through a call to a custom notification rule.
12. The workflow process for the event ends.
API: Depending on whether the event is approved or rejected, Identity Manager calls the `approved()` or `rejected()` method in each event listener for the event `ModifyUserEvent`.
13. Identity Manager sends email notification about the event result.
API: An email notification can include event-specific information through JavaScript calls to the Email Template API. Also, email recipients can include users who are specified through a call to a custom notification rule.
14. If the event is approved, Identity Manager executes the event.
API: After executing the event, Identity Manager calls the `after()` method in each event listener for the event `ModifyUserEvent`.
If an event listener is performing rule-based role or group assignments, it is typically at this point that the assignments are made.
15. If the event is approved, Step 9 through Step 14 for the event `AddToGroupEvent` are performed.
In this example, `ModifyUserEvent` is a primary event. If a primary event is rejected, no other events are processed (in this example, `AddToGroupEvent`), and the Modify User Task is not performed.
16. After the workflow process for each event is completed, and if the event `ModifyUserEvent` was approved, Identity Manager completes the Modify User task.
17. Identity Manager sends email notification about the task result.
API: An email notification can include event-specific information through JavaScript calls to the Email Template API. Also, email recipients can include users who are specified through a call to a custom notification rule.

API Functional Model

The design and implementation of all Identity Manager APIs are based on a common functional model.

There are two variations of this model, depending on whether task execution is in the synchronous phase (before the task has been broken down into events) or the asynchronous phase (after the task has been broken down into events).

This section contains an illustration of each variation of the functional model, and also a summary of all the components of the functional model.

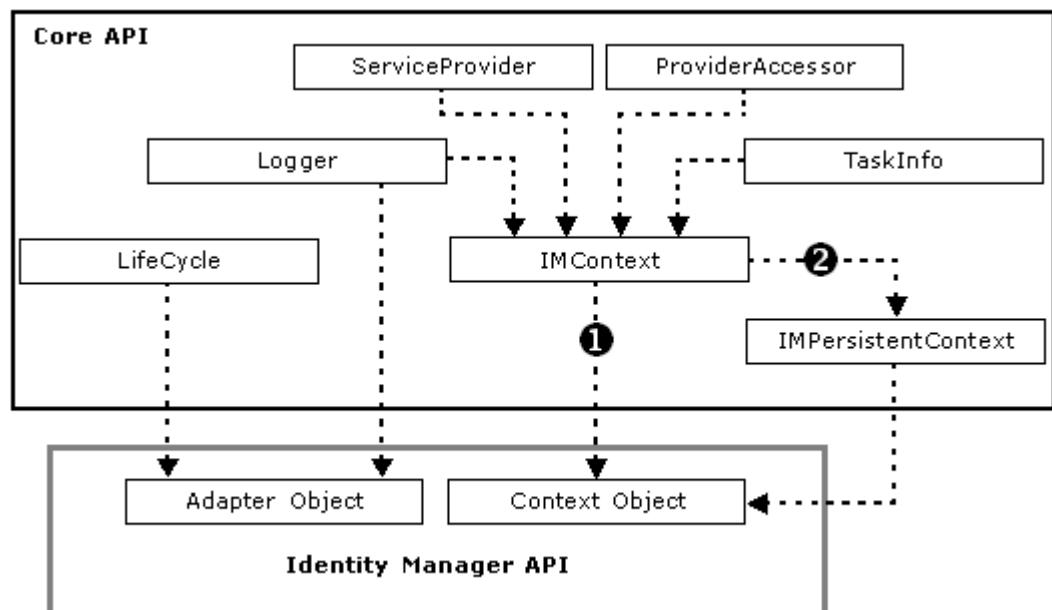
Synchronous Phase Model

In the synchronous phase of task execution, the task has not yet been broken down into its component events.

The following Identity Manager APIs can be used during this phase. Each API inherits methods from the core IMContext interface in different ways:

- **Logical Attribute API**—Inherits IMContext methods directly (alternative 1 in the following diagram).
- **Business Logic Task Handler API**—Inherits IMContext methods through the IMPersistentContext interface (alternative 2 in the diagram).

The following diagram illustrates the core classes and interfaces in the synchronous phase model, and shows where an individual API fits into the model:



Asynchronous Phase Model

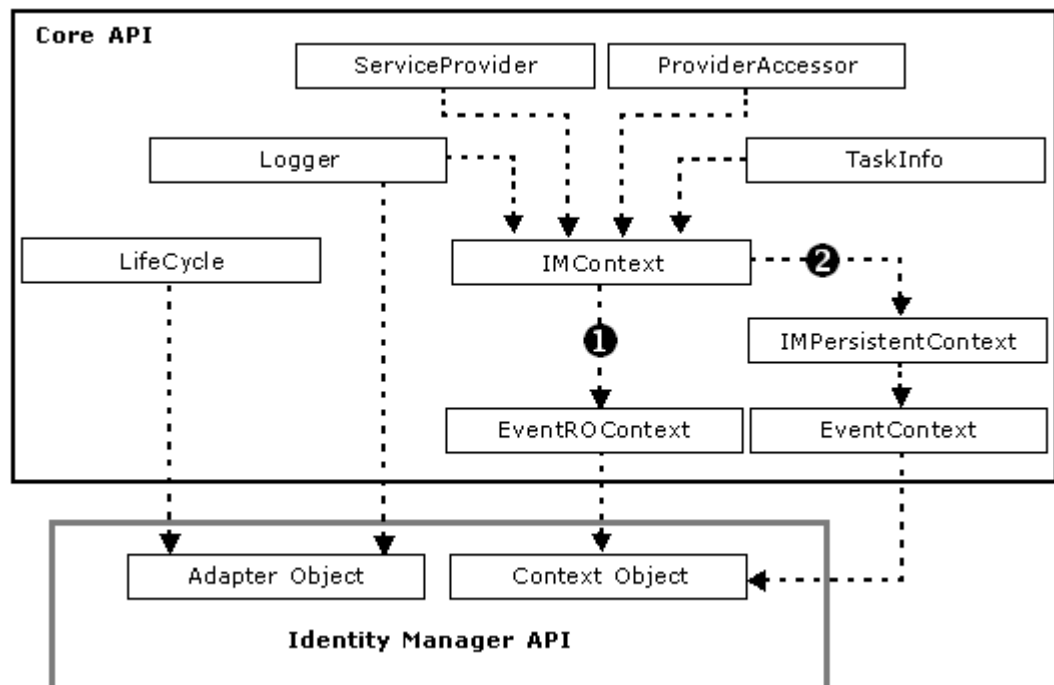
In the asynchronous phase of task execution, the task has been broken down into its component events. As a result, the core objects in the API functional model now include:

- **EventContext**—Provides read/write access to events and lets you generate secondary events.
- **EventROContext**—Provides read/write access to events.

The following Identity Manager APIs can be used during this phase. The APIs inherit methods from the core IMContext interface in different ways:

- **Participant Resolver API** and **Notification Rule API**—Inherit IMContext methods through EventROContext (alternative 1 in the following diagram).
- **Event Listener API** and **Workflow API**—Inherit IMContext methods through the IMPersistentContext and EventContext interfaces (alternative 2 in the diagram).

The following diagram illustrates the core classes and interfaces in the asynchronous phase model, and shows where an individual API fits into the model:



Model Components

The API functional model consists of the following core components:

- **LifeCycle interface**—Provides custom objects with access to user-defined properties defined in the Identity Manager Management Console. The properties can be applicable to a single custom object and/or to the entire Identity Manager environment.

Also provides a custom object with basic implementations for startup and shutdown methods, and optionally, access to the managed object providers.

LifeCycle methods are called when Identity Manager starts up the environment and loads objects (including custom objects), and also when the environment is shut down. These methods give the custom object an opportunity to connect to and disconnect from external objects, and to perform any other startup and shutdown operations that it might require.

The base adapter classes for the individual Identity Manager APIs implement the LifeCycle interface.

- **ServiceProvider interface**—Provides Identity Manager APIs with access to services such as password and encryption operations.
- **ProviderAccessor interface**—Makes the Identity Manager providers available to custom objects. The providers allow direct access to managed object data.
- **TaskInfo interface**—Provides information about the current task—for example, task creation time, the name and unique identifier of the administrator who is executing the task, and the organization where the task is being executed.
- **Logger interface**—Lets a custom object log messages directly to the application server log.
- **IMContext interface**—Extends the TaskInfo, ServiceProvider, ProviderAccessor, and Logger interfaces, and makes the methods in these interfaces available to the ...Context objects in the individual APIs.

- **IMPersistentContext interface**—Allows custom objects to persist user-defined data within a task session, as follows:
 - Between business logic task handlers for a given task (Business Logic Task Handler API).
 - Across different states of an event (Event Listener API).
 - Between different scripts for a given workflow activity (Workflow API).
 - Between different types of custom objects for a given task (various APIs).

For example, user-defined data can be set by a business logic task handler during the synchronous phase of task processing, and then retrieved by an event listener during the asynchronous phase of the same task session (Business Logic Task Handler API, Event Listener API).

This interface extends IMContext.

Note: The persisted user-defined data is defined within the IMPersistentContext object. It is not the same user-defined data that is defined in the Identity Manager Management Console for a particular custom object or for the Identity Manager environment.

- **EventContext interface**—Provides read/write access to the managed object attributes in an event, and lets you generate secondary events. APIs that can implement this interface are the Event Listener API and Workflow API.

This interface extends IMPersistentContext, which extends IMContext. This interface also serves as the base interface for context objects in the Event Listener API.

- **EventROContext interface**—Provides read/write access to events. APIs that can implement this interface are the Participant Resolver API and Notification Rule API.

This interface extends IMContext.

The individual Identity Manager APIs typically have the following types of components:

- **Adapter object**—Each Identity Manager API has a base class called an adapter. The API-specific adapter implements the LifeCycle and Logger interfaces.

The custom objects you create with the Identity Manager APIs extend this base class. An adapter object contains base implementation methods inherited from LifeCycle for startup and shutdown operations, and also API-specific methods.

- **Context object**—Each Identity Manager API has an API-specific context object, which is passed into the API's adapter object. The context object inherits core methods through IMContext, either directly or indirectly. Consequently, the API's context object contains information about the current task, and also provides access to logging methods and other Identity Manager services, including managed object retrieval from the data store.

Note: For information about these components and all Identity Manager APIs, see the Javadoc Reference.

Chapter 2: Logical Attribute API

This section contains the following topics:

[Logical Attributes and Physical Attributes](#) (see page 31)

[Attribute Data Flow](#) (see page 32)

[Logical Representation of Physical Attributes](#) (see page 33)

[Logical Attribute Elements](#) (see page 34)

[Predefined Logical Attributes](#) (see page 35)

[Logical Attribute API Overview](#) (see page 38)

[Logical Attribute Configuration](#) (see page 44)

[Option Lists](#) (see page 45)

[Self-Registration Logical Attribute Handler](#) (see page 48)

[Logical Attribute Handler for a Forgotten Password](#) (see page 51)

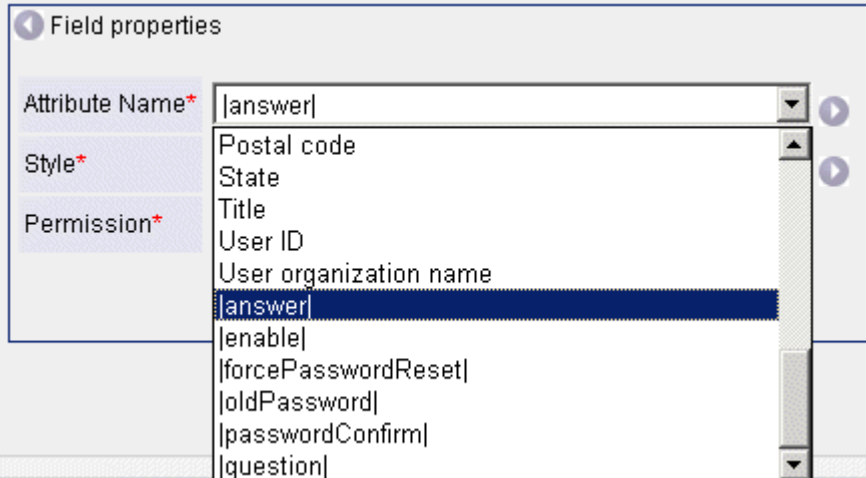
[Logical Attribute Implementation](#) (see page 53)

[Screen-Defined Logical Attributes](#) (see page 58)

Logical Attributes and Physical Attributes

Task screen fields are associated with data attributes. For example, on a user profile screen, the task screen fields First Name and Last Name might be associated with the attributes *firstname* and *lastname*, respectively, in the underlying user directory.

The attributes that can be associated with the fields on a particular task screen appear in the Attribute Name drop-down for that task screen, as shown in the following list:



The screenshot shows a 'Field properties' dialog box with a table-like structure. The first column contains labels with red asterisks: 'Attribute Name*', 'Style*', and 'Permission*'. The second column contains a list of attribute names. The 'Attribute Name' row is expanded, showing a scrollable list of options: 'Postal code', 'State', 'Title', 'User ID', 'User organization name', '|answer|' (which is highlighted in blue), '|enable|', '|forcePasswordReset|', '|oldPassword|', '|passwordConfirm|', and '|question|'. There are small circular icons with arrows on the right side of the dialog box.

Attribute Name*	Style*	Permission*
Postal code		
State		
Title		
User ID		
User organization name		
answer		
enable		
forcePasswordReset		
oldPassword		
passwordConfirm		
question		

Attribute Name contains two types of data attributes:

Physical attribute

Physical attributes are attributes of the underlying data store (a user directory such as LDAP, the SiteMinder policy store, or a database). If a task screen field is configured with a physical attribute, a value entered into the field is ultimately written to the data store.

For example a Last Name field on a user profile screen might be configured with a *lastname* attribute in the underlying user directory. When a user enters a value into the field and clicks Submit, the value is written to the *lastname* attribute in the directory. When a user next displays the profile screen, the value is retrieved from the *lastname* attribute of the directory and displayed in the Last Name field.

Logical attribute

Logical attribute values are not directly associated with or written to the data store. A logical attribute value is presented in a task screen field and processed by a logical attribute handler.

The result of the processing may be written to the data store if the logical attribute is mapped to a physical attribute, or the result may be used in some other way. For example, a logical attribute handler might evaluate the contents of a Notify me of good deals field to determine whether a self-registering user should be added to an organization whose users receive promotional materials and other benefits.

More Information:

[Logical Attribute Handler at Run Time](#) (see page 54)

Attribute Data Flow

Data flow is the transfer of attribute data to and from a task screen. Data is written to the task screen just before the task screen is displayed, and read from the task screen when the user executes (submits) the task.

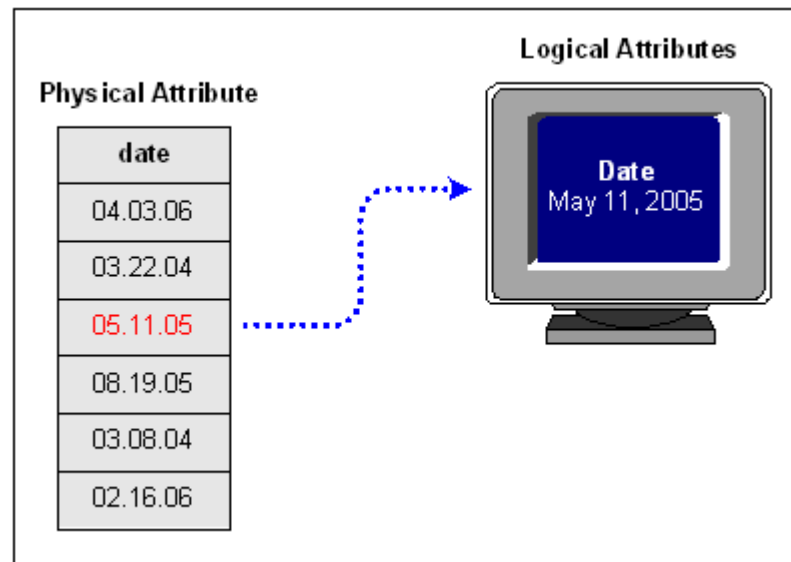
In a simple data flow, a task screen field is associated with a physical attribute. Data in this field flows to and from the corresponding physical attribute in the data store. In this case, no logical attribute handlers are involved in the data flow between the task screen and the data store.

If a task screen field is associated with a logical attribute rather than a physical attribute, data flows between the field and a logical attribute handler, allowing the handler to process the data. Optionally, the logical attribute handler can convert the logical attribute data to physical attribute data that can be written to the data store.

Channelling data flow through a logical attribute handler allows customization opportunities such as the following:

- Controlling the way physical data is presented on the task screen.
- Validating data that users supply on the task screen.
- Integrating your enterprise's business logic and rules with the presentation that users see.

For example, logical attributes let you to present data on a task screen in a more user-friendly format than the corresponding physical attribute data in the data store, as shown:



Logical Representation of Physical Attributes

Logical attributes and physical attributes do not necessarily correspond on a one-to-one basis. Logical attributes can represent physical attributes in the following ways:

- One logical attribute representing one physical attribute, as in the case of task screen input being stored in coded form. For example, an *AccountType* attribute might contain a value of Gold, Silver, or Bronze on the task screen, but 1, 2, or 3, respectively, in the data store.
- One logical attribute representing multiple physical attributes. For example, the logical attribute *Cost* may represent the product of the physical attributes *UnitPrice* and *Quantity*.

- Multiple logical attributes representing one physical attribute. For example, a password challenge question and answer can be stored as one delimited physical attribute, but represented by two logical attributes, *question* and *answer*.
- A logical attribute with no corresponding physical attribute. For example, a user who self-registers can enter a code to gain access to specific information, such as special promotions. In this case, the user-supplied code is a logical attribute that tells Identity Manager the organization in which to create the user's profile. The code is not written to the data store.

Logical Attribute Elements

Logical attribute functionality requires an association of elements such as the following:

- The name of the logical attribute handler that processes the logical attribute data.

- The name of one or more logical attributes. Logical attribute names are enclosed in vertical bars (|), for example:

|answer|

Typically, each logical attribute is assigned to a task screen field. Logical attribute names that can be assigned to task screen fields are listed in the Attribute Name drop-down list.

- The name of the corresponding physical attributes, if any.

Physical attributes are defined in the <ImsManagedObjectAttr> element of directory.xml. <ImsManagedObjectAttr> maps physical attribute names to the following names:

- Display names (field displayname). In a task screen, a physical attribute is represented in the Attribute Name drop-down list by its display name.
- Optionally, a well-known attribute name (field wellknown). Well-known attribute names are enclosed in percent signs (%), for example:
%ENABLED_STATE%

Because a well-known attribute is mapped to a physical attribute, a logical attribute can be mapped to a physical attribute indirectly, through a well-known attribute.

- The type of managed object (USER, ORG, or GROUP) that the logical attribute applies to.

When a task of the specified object type is being defined, logical attributes associated with the object type appear in the task screen's Attribute Name dropdown, and therefore can be assigned to a task screen field.

More Information:

[Logical Attributes and Physical Attributes](#) (see page 31)

[Logical Attribute Handlers for Option Lists](#) (see page 45)

Predefined Logical Attributes

Identity Manager includes predefined logical attributes. You can use a predefined logical attribute as is, or customize it to suit your business requirements.

The predefined logical attributes are described here:

Password

Logical Attribute

|passwordConfirm|

|oldPassword|

Physical Attribute

Display name: Password

Well-known name: %PASSWORD%

Description

Processed by the Confirm Password Handler.

When a user specifies a new password, the password is validated against the password supplied in the confirmation field to make sure the values match.

Typically, *oldPassword* is defined as a hidden field, while *Password* and *passwordConfirm* are displayed.

Available for USER objects.

Password Hint

Logical Attribute

|Question1|... |Question5|

|Answer1|... |Answer5|

|Questions|

|VerifyQuestion|

|VerifyAnswer|

Physical Attribute

Well-known name: %PASSWORD_HINT% (a multi-value attribute)

For eTrustAdmin: eTSelfAuth-Question0-4
eTSelfAuth-Answer0-4

Description

Processed by the Forgotten Password Handler.

Questions and answers for identification can be stored in the directory in various ways, depending upon the configuration. Optionally, the user can be challenged with a verification question at run time. There can be more than one verification question and answer.

Password Reset

Logical Attribute

|forcePasswordReset|

Physical Attribute

Display name: Disabled State

Well-known name: %ENABLED_STATE%

Description

Processed by the Forced Password Reset Handler.

If the administrator enables *forcePasswordReset* on a task screen, the user's stored enabled state is updated so that the user must change his password during his next login.

Available for USER objects.

Enable User

Logical Attribute

|enable|

Physical Attribute

Display name:
Disabled State

Well-known name:
%ENABLED_STATE%

Description

Processed by the Enable User Handler.

Both of these actions:

- If the administrator enables or disables a user's account through the *enable* attribute, the user's stored enabled state is updated accordingly.
- Before the task screen is displayed, the user's stored enabled state value is retrieved from the directory and converted to the logical attribute *enable*.

Note: If this logical attribute is not used or is empty, the user's account is enabled by default.

Available for USER objects.

Group Subscription

Logical Attribute

|GroupSubscription|

Physical Attribute

Display name: Self Subscribing

Well-known name: %SELF_SUBSCRIBING%

Description

Processed by the Self Subscribing Handler.

All of these actions:

- If the administrator sets this attribute to TRUE, users can add themselves to the group.
- Converts between physical and logical attribute values.
- Validates the self-subscription value before the value is written to the data store.

Available for GROUP objects.

More Information:

[Logical Attribute Handler for a Forgotten Password](#) (see page 51)

Logical Attribute API Overview

The Logical Attribute API lets you create and manage custom attributes that can be associated with task screen fields.

You use the Logical Attribute API to write a handler object that performs operations such as the following:

- Formatting logical attribute values displayed on a task screen.
- Populating fields configured with logical attributes when a task screen is displayed.
- Validating user-supplied values in fields configured with logical attributes.
To perform broader validation operations against information in the entire task, use the Business Logic Task Handler API.
- Adding to a task screen a multi-value field (a field that can have more than one value) or an option list (a drop-down list containing selectable values).
- Processing logical attribute values or associated physical attribute values to suit your business requirements.
- Converting user-supplied logical attribute values to physical attribute values after the user submits the task for processing. Identity Manager then writes the physical attribute values to the data store.

Logical Attribute API Summary

Operations

- Identity Manager Displays and Populates the Task Screen.
- Identity Manager Invokes Validation Rules During Data Entry.
- User Submits the Task Screen.

Called by

Identity Manager.

When called

Any of these circumstances:

- After the user selects a task to perform, and just before the task screen appears.
- The user clicks Submit on a task screen.

Operates on

Logical attributes and physical attributes associated with the logical attribute handler.

Object update

Logical attributes and physical attributes associated with the logical attribute handler.

Can validate?

Logical attributes and physical attributes associated with the logical attribute handler.

Note: Logical attribute handler execution occurs in the synchronous phase of task processing. If response time during interaction with a third-party system is an issue, the same objective can be performed using an event listener during the Pre-approval state.

Note: You can also perform attribute validation using JavaScript and regular expressions. For information, see the *CA Identity Manager Configuration Guide*.

Use Case Examples

Examples of how the logical attribute API can be used:

Format data from a data source

In the user directory, a telephone number is stored as the physical attribute *phone*. It has 10 digits—for example, 9785551212. This physical attribute is associated with the logical attribute *formatted_phone*.

When the task screen that contains the phone number is about to be displayed, Identity Manager calls a custom logical attribute handler that formats the physical attribute phone number appropriately—for example, as (978) 555-1212—and assigns the formatted value to *formatted_phone*. Identity Manager uses the *formatted_phone* logical attribute value to display on the task screen.

Validate data entry

The physical attribute *employeeNumber* exists in a data source. It is mapped to the logical attribute *validEmployeeNumber*. After a user assigns an employee number to the *validEmployeeNumber* attribute on the task screen and clicks Submit, Identity Manager calls a custom logical attribute handler to query the data source to ensure that the supplied employee number exists in *employeeNumber*. If it does not exist, the logical attribute handler throws an exception.

Add a drop-down list to a task screen

A task screen requires a customer to provide the type of service he has purchased—Gold, Silver, or Bronze. These options are presented in a dropdown list associated with the logical attribute *serviceType*. When the customer chooses a service level and clicks Submit, Identity Manager calls a logical attribute handler that converts the selected service level to the corresponding physical attribute value (1, 2, or 3) and assigns the value to the physical attribute *type*.

More Information:

[Use Case Example: Formatting a Date Attribute](#) (see page 55)

API Components

The Logical Attribute API contains the following components:

LogicalAttributeAdapter

The base class that all logical attribute handlers extend. Contains base implementation methods for startup and shutdown operations. Also performs operations such as validation of logical attribute values and conversion between logical attribute values and physical attribute values. These operations are based on the LogicalAttributeContext information passed into the handler.

Implements Lifecycle and Logger.

LogicalAttributeContext

Interface providing read and write access to the logical attributes and any associated physical attributes for a particular logical attribute scheme.

This interface also provides access to IMContext information about the current task, and to logging methods and other Identity Manager services, including managed object retrieval from the data store.

LogicalAttributeContext information is passed to the methods in the LogicalAttributeAdapter object.

Extends IMPersistentContext.

IMPersistentContext

Allows user-defined data to be passed between a logical attribute executed for a given task session, and between logical attribute handlers and other custom objects in the same task session. The persisted data is abandoned after the task session ends.

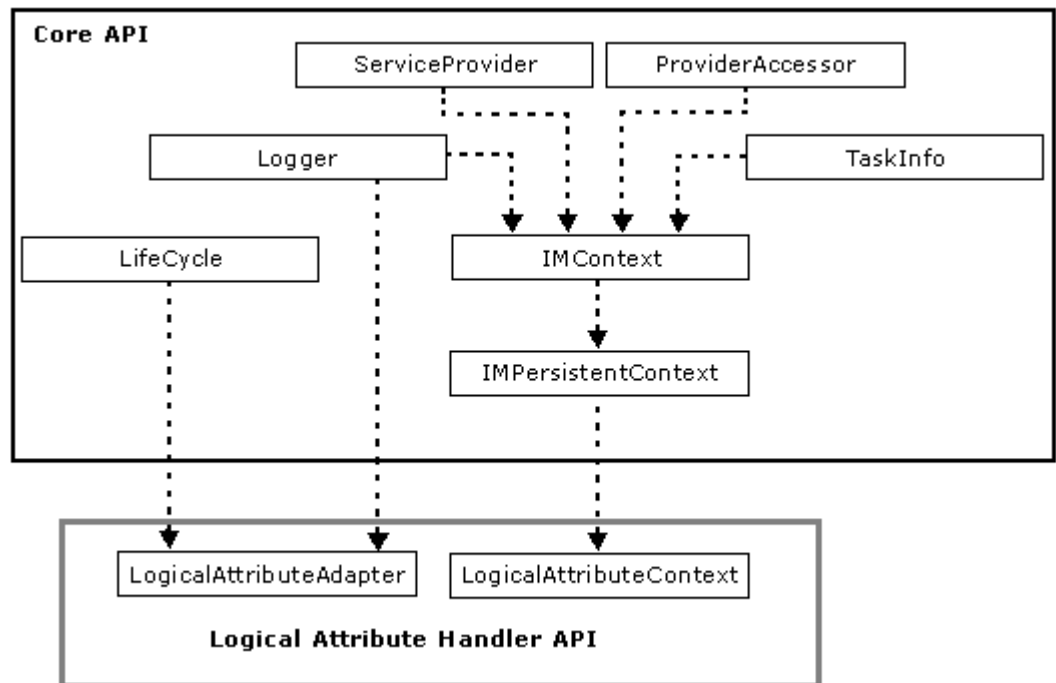
This interface is part of the core API functional model.

Extends IMContext.

The above components are in the package com.netegrity.imapi.

Logical Attribute Handler API Model

The components of the Logical Attribute API are part of the Identity Manager API functional model:



Calling Sequence

As with all custom objects and handlers created with the Identity Manager APIs, Identity Manager executes a logical attribute handler by calling methods in the handler's adapter object.

Identity Manager calls LogicalAttributeAdapter methods at different stages of task screen processing:

1. initializeOptionList(). This is the first method Identity Manager calls before it displays a task screen. This method gives the logical attribute handler an opportunity to populate any option lists on the task screen.
2. Either of the following methods:
 - toLogical(). For tasks (such as Modify User) that act on existing managed objects, allows the logical attribute handler to convert the managed object's physical attribute values to logical attribute values.
 - initialize(). For tasks that create a managed object (such as Create User), allows the logical attribute handler to supply default logical attribute values for display on the task screen.

When toLogical() or initialize() returns successfully, Identity Manager displays the task screen.

3. validate(). Allows the handler to validate user input. This method is called after the user clicks Submit on the task screen.
4. toPhysical(). Allows the handler to convert logical attribute values to physical attribute values.

Identity Manager calls logical attribute handlers in the order in which they are listed in the Identity Manager Management Console. In each successive handler, Identity Manager calls the method that is appropriate for a given stage of task screen processing. For example, validate() is called in each handler when the user clicks Submit.

Note: Identity Manager calls business logic task handlers after all logical attribute handler processing is complete. Thus, business logic task handlers have access to the logical attribute values and the physical attribute values set by the logical attribute handlers.

Custom Messages

After a user submits a task, Identity Manager can display the following user-defined messages:

Informational messages

A logical attribute handler can define a custom informational message, such as an acknowledgement, to display to the user after the user successfully submits a task.

To display a custom informational message for the task, specify the message in an `addMessageObject()` method inherited by `LogicalAttributeContext`.

Exception messages

If validation errors occur after a user submits a task, the task screen is redisplayed with the appropriate exception messages. This gives the user the opportunity to correct the errors and resubmit the task.

To enable Identity Manager to display custom exception messages on a task screen, add the messages to an `IMSEException` object.

Identity Manager supports localized exception messages, which you store in a family of resource bundles. Each resource bundle in the family contains the same messages, but in a different language. You specify the current user's locale and the base name of the resource bundles, and Identity Manager selects the appropriate resource bundle from which to retrieve exception messages.

For detailed information, see the `IMSEException` class in the Javadoc reference.

More Information:

[IMSEException](#) (see page 134)

Sample Logical Attribute Handlers

Sample implementations of logical attribute handlers are installed in the `samples\LogicalAttributes` subdirectory of your Identity Manager installation—for example:

`C:\Program Files\CA\Identity Manager\samples\LogicalAttributes`

Note: For information about using the samples, see the `readme.txt` files that accompany the samples.

Logical Attribute Configuration

You define a logical attribute handler's fully qualified class name and other properties in the Identity Manager Management Console.

To define a logical attribute handler:

1. In the Management Console, click Identity Manager environments.
2. Click the name of the environment where the logical attribute handler will be used.
3. Click Advanced Settings.
4. Click Logical Attribute Handlers.

The Logical Attribute Handlers screen contains a list of existing logical attribute handlers. The list includes predefined handlers shipped with Identity Manager and any custom handlers defined at your site. Identity Manager executes the handlers in the order in which they appear in this list.

From the Logical Attribute Handlers screen, you can do the following:

- Register a new logical attribute handler with Identity Manager. To begin this operation, click New.
- Modify an existing handler configuration. To begin this operation, click the handler name.
- Delete a handler.
- Change the order of handler execution.

For detailed instructions about performing these operations, click the Help link on the Logical Attribute Handlers screen.

Note: You can also create, modify, view, and delete a logical attribute handler from the User Console through the Environment task category. See the online Help for further information.

Place the handler in the package location specified in the Identity Manager Management Console.

More Information:

[Compiling and Deploying](#) (see page 145)

Option Lists

An option list allows a user to assign values to a field by selecting one or more items in a list rather than typing values into a text box.

You create an option list using the following Identity Manager features:

Logical Attribute API

Lets you populate the option list, convert values between the associated logical attribute and physical attribute, and validate the selected values.

Identity Manager Management Console

Lets you configure the logical attribute handler that populates the option list and processes the selected values.

Identity Manager User Console

Lets you add an option list to a task screen.

Populating Option Lists

An option list is populated programmatically through a custom object that you create with the Logical Attribute API. The logic is performed in the `initializeOptionList()` method of the `LogicalAttributeAdapter` class. You can populate the option list through whatever mechanism you like—for example, from hard-coded items, from items in a data source, or from items in a text file.

Identity Manager calls `initializeOptionList()` after a user requests an admin task and the task screen is about to be displayed.

Logical Attribute Handlers for Option Lists

You configure the logical attribute handler that populates the option list in the Identity Manager Management Console. When you do so, make sure that you complete the following tasks:

- Define the following two logical attributes:
 - One logical attribute to contain the values in the option list.
In the handler, use the method `initializeOptionList()` to populate the list items.

- One logical attribute to display the values currently selected in the list.

This logical attribute is like any other logical attribute you add to a task screen. It may be associated with one or more physical attributes, and it may be restricted to a single value or allow multiple values.

In the handler, use the methods `toPhysical()` and `toLogical()` to convert values to and from this logical attribute and any associated physical attribute.

- Typically, you define a physical attribute that corresponds to the attribute that displays the selected values. This physical attribute/logical attribute relationship is like any other that does not involve an option list.
- Optionally, specify user-defined properties that the handler uses when populating the option list and processing the selected values. For example, you might pass in the name of a file that contains the items in the option list.

Adding an Option List to a Task Screen

An option list is a type of field you can add to a task screen, just like a text field or other field type. You add the option list when you are defining a task screen in the Identity Manager User Console.

The only attribute you need to add to the task screen is the logical attribute that displays the selected values. Identity Manager automatically manages the associated physical attribute and the logical attribute that contains the list of values. These logical and physical attributes are not displayed to the end user.

To add an option list to a task screen field in the User Console:

1. Make sure the field to be configured with the logical attribute is included on the task screen.
2. Access Field properties for the field.
3. Select the name of the logical attribute that will display the selected values.
4. Assign one of the following style types to the field:
 - Single-Select. Allows users to select a single value from a list box.
 - Multi-Select. Allows users to select multiple values from a list box. This style type must be associated with a multi-valued attribute.
 - Dropdown. Allows users to select one or more values from a dropdown list.

More Information:

[Example: Assigning a Logical Attribute to a Field](#) (see page 54)

Option List Configuration Summary

The following table summarizes the basic configuration properties for logical and physical attributes associated with an option list.

You set these properties in the Identity Manager User Console and the Identity Manager Management Console.

Attribute Type	Purpose	User Console Style Type	Management Console Properties
Logical attribute	Contains one selected value	Single Select or Drop-down	Name and Attribute Name
Logical attribute	Contains one or more selected values	Multi-Select	Name, Attribute Name, and Multivalued
Logical attribute	Contains the list of items	—	Name, Attribute Name, and Option List
Physical attribute	Corresponds to the logical attribute that contains the selected values	—	Name and Attribute Name

Option Lists at Run Time

When an option list is properly configured and the task screen is displayed to an end user at run time:

- Identity Manager fills the list portion of the logical attribute with the list items defined through `initializeOptionList()` in the logical attribute handler.
- Identity Manager manages the appropriate display and removal of the list.
- When the user selects an item in the list, Identity Manager automatically displays the selected item in the field configured with the logical attribute.

Sample Option List

A sample logical attribute handler that populates an option list and processes selected values is provided with Identity Manager. The sample includes a readme file that describes how to configure the option list.

The sample is installed in the samples\LogicalAttributes\StateSelector subdirectory of your Identity Manager installation—for example:
C:\Program Files\CA\Identity Manager\samples\LogicalAttributes\StateSelector

Self-Registration Logical Attribute Handler

When a user self-registers, the user's organization can be specified in the following ways:

- Through a default organization for self-registered users.
- Through custom code that determines the user's organization by analyzing data that the user provided on the self-registration screen. This approach is based on logical attribute functionality.

Before reading these sections, you should familiarize yourself with configuring logical attributes.

Note: For information about specifying the user's organization, see the *CA Identity Manager Administration Guide*.

More Information:

[Logical Attribute Configuration](#) (see page 44)

How a Self-Registration Handler Works

A self-registration handler is very similar to a typical logical attribute handler:

- You use the Logical Attribute API to write the custom handler. The custom handler extends the OrgSelectorAdapter abstract class, which extends LogicalAttributeAdapter.
- You define an Organization Selector (self-registration handler) in the Identity Manager Management Console for your Identity Manager environment. The property definitions map the self-registration handler class file to one or more logical attribute names.
- You add the logical attributes defined in the Identity Manager Management Console to your self-registration screen.

- Identity Manager calls out to the self-registration handler when the user submits the self-registration screen containing the logical attributes.
- The self-registration handler processes the information that Identity Manager passes to it (which includes the logical attribute values entered on the task screen), determines the organization where the user should be added, and passes the organization back to Identity Manager.

There are also important differences:

- The OrgSelectorAdapter class is only used with self-registration handlers.
- Identity Manager calls out to the self-registration handler when the user changes tabs during self-registration. Logical attribute handlers are not invoked when tabs are changed.
- Logical attributes are often associated with physical attributes. But a logical attribute mapped to a self-registration handler is typically not associated with a physical attribute.

Writing the Self-Registration Handler

You write the custom handler code by extending the OrgSelectorAdapter abstract class. This class extends LogicalAttributeAdapter.

OrgSelectorAdapter has one API-specific (non-inherited) method—getOrganization(). When the user submits the self-registration screen, Identity Manager calls getOrganization() and passes the following objects to it:

- A populated LogicalAttributeContext object. This object includes access to logical attributes on the task screen.
- A Vector of organizations where the self-registering user can be added. The getOrganization() method returns one of these organizations.

The getOrganization() method evaluates the logical attribute values supplied on the task screen and determines the organization where the user should be added. For example, a user's organization might be determined by the job description information provided on the self-registration screen. The method determines the user's organization and returns the organization object to Identity Manager.

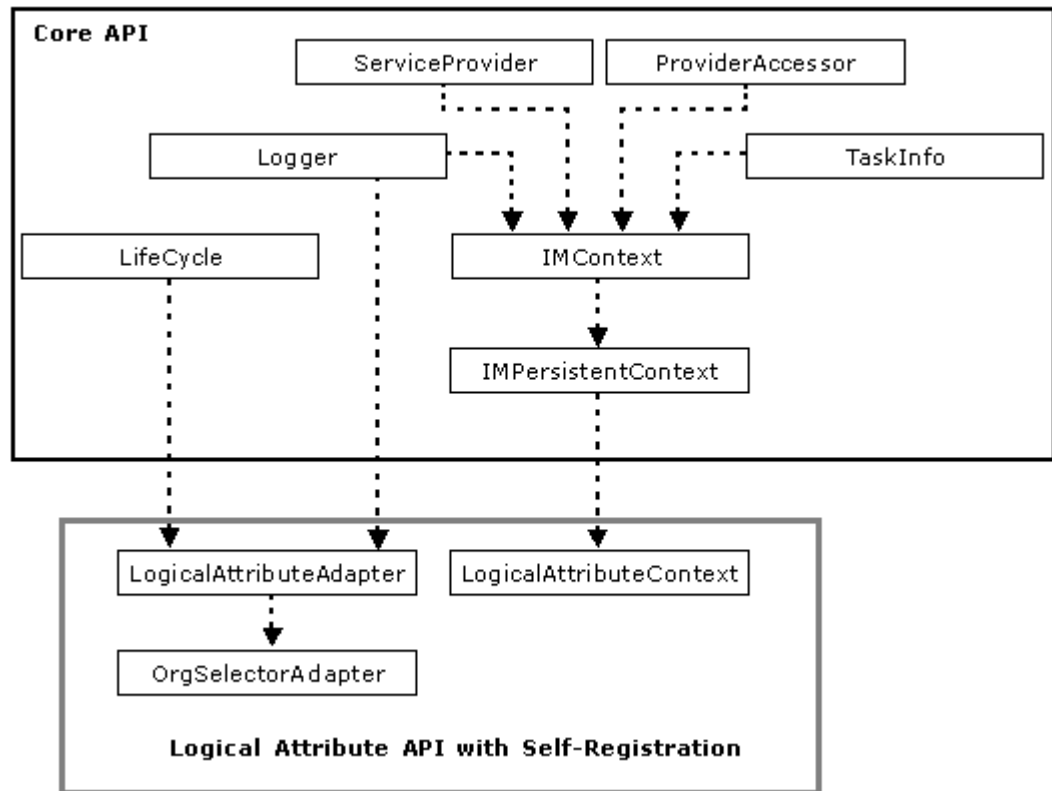
Note: Because OrgSelectorAdapter extends the abstract class LogicalAttributeAdapter, your derived class must declare the methods in LogicalAttributeAdapter even if the methods remain empty.

More Information:

[Logical Attribute API Summary](#) (see page 38)

Self-Registration and the API Functional Model

A self-registration handler uses the same core components of the Identity Manager API functional model as the Logical Attribute API. In addition, the self-registration class `OrgSelectorAdapter` extends `LogicalAttributeAdapter`:



Self-Registration Handler Configuration

You define a self-registration handler's fully qualified class name, logical attribute name or names, and other properties in the Identity Manager Management Console.

To define a self-registration handler:

1. In the Management Console, click Identity Manager Environments.
2. Click the name of the environment where the self-registration handler will be used.
3. Click Advanced Settings.
4. Click Organization Selector.

On the Organization Selector screen, you can:

- Register a new self-registration handler.
- Modify the self-registration handler's properties if the handler already exists.

For detailed instructions about performing these operations, click the Help link on the Organization Selector screen.

Note: There can be only one self-registration handler per Identity Manager environment. Consequently, there is only one set of Organization Selector properties that can be displayed. Also, there is no Name property for the Organization Selector.

Sample Self-Registration Handler

A sample implementation of a self-registration handler is installed in the samples\LogicalAttributes\OrgSelector subdirectory of your Identity Manager installation—for example:

C:\Program Files\CA\Identity Manager\samples\LogicalAttributes\OrgSelector

For information about using the sample, see the readme.txt file that accompanies the sample.

Logical Attribute Handler for a Forgotten Password

Identity Manager provides a Forgotten Password task that lets you reset or retrieve a password. You can respond to one or more verification questions at run time, or to choose questions from a predefined list.

When you initiate the Forgotten Password task, Identity Manager takes the following steps:

1. Identity Manager displays an Identify screen. The user enters an ID.
2. The Forgotten Password Search Handler looks up the user. If the user is found, the task enters the verification stage. If no user is found, an error message is displayed, and the user can try again. An administrator can configure limits on failed or successful attempts.
3. For verification, the Forgotten Password Search Handler invokes the Forgotten Password Handler, which converts the defined physical attributes to logical ones.
4. The Forgotten Password handler also presents at least one verification question. The Forgotten Password Search Handler validates the answer and checks for success based on task configuration.

5. The Forgotten Password Search Handler invokes the Forgotten Password Handler as many times as necessary if more verification questions are configured.
6. After the criteria for success have been met, control passes to the tabs configured for the task. Depending on the configuration, the user can reset the password. Alternatively, Identity Manager can issue a temporary password, which is normally displayed, or can be sent by email to the user.

Writing a Custom Forgotten Password Handler

You can modify the functionality of the Forgotten Password Handler in two ways:

- You can implement a different manner of retrieving the verification questions and answers.
- You can implement your own scheme for encrypting the questions and answers.

The Forgotten Password Handler defines an API-specific method, `getQuestions()`, that creates a Vector of verification questions. The questions and answers are passed to a method for presentation to the user. The `getQuestions()` method is called by the handler's `init()` method. Its input is a hashtable of values configured by an administrator through the Forgotten Password task.

Note: For information about how the Forgotten Password task are configured, see the *CA Identity Manager Configuration Guide*.

You might, for example, have a setup where your questions and answers are stored in a database and you do not want to update an Identity Manager property file every time the database is updated. You could extend the `ForgottenPasswordHandler` class and implement the `getQuestions()` method to fetch the questions from the database on the next initialization of your handler.

Another possibility for extending this class is to override the `encrypt()` and `decrypt()` methods. The `init()` method gets the encryption key if one has been configured. By default, no key is provided. The `encrypt()` method is called from the `toPhysical()` to convert the names and addresses before they are written out to the user store. The `decrypt()` method is called by `toLogical()` to convert these values when they are read from the user store. You can implement the `encrypt()` and `decrypt()` methods to perform any encryption that is appropriate for your installation.

Note: When using the "Separate Attributes" schema for questions and answers (like eTrust Admin), a separate attribute is required in order to enable control data. The correct exception is displayed when you execute Forgotten Password.

Forgotten Password Handler Configuration

You define a Forgotten Password Handler's fully qualified class name, logical attribute name or names, and other properties in the Identity Manager Management Console.

To define a Forgotten Password Handler:

1. In the Management Console, click Identity Manager Environments.
2. Click the name of the environment where the forgotten password handler will be used.
3. Click Advanced Settings.
4. Click Logical Attribute Handlers.
5. Click Forgotten Password Handler.

On the Forgotten Password Handler screen, you can do the following:

- Register a new forgotten password handler.
- Modify the forgotten password handler's properties if the handler already exists.

For detailed instructions about performing these operations, click the Help link on the Forgotten Password screen.

Logical Attribute Implementation

The table following summarizes the high-level actions that must be performed when you create a logical attribute, and specifies where you can look for detailed information about an action.

Although the actions are presented in a logical sequential order, some steps can be performed in a different order.

1. Write the logical attribute handler code.
2. Register the logical attribute handler in the Identity Manager Management Console for the Identity Manager environment.
3. Compile and deploy the handler.
4. Add the logical attribute to a task screen being constructed in the Identity Manager User Console.

More Information:

[Logical Attribute API Summary](#) (see page 38)

[Logical Attribute Handlers for Option Lists](#) (see page 45)

[Compiling and Deploying](#) (see page 145)

[Logical Attribute Handler at Run Time](#) (see page 54)

Logical Attribute Handler at Run Time

After you create and deploy the logical attribute handler, register the new logical attribute handler in the Identity Manager Management Console, and restart the application server, the newly created logical attribute is ready for use. You can now assign the logical attribute name to a task screen field. Assigning a logical attribute to a field is no different than assigning a physical attribute to a field.

During task screen configuration, logical attributes that can be assigned to task screen fields appear in the Attribute Name dropdown list. Attribute Name is one of the field properties you define when configuring a particular field on the task screen. Identity Manager includes registered logical attribute names in this dropdown list automatically.

Both logical attribute names and physical attribute names are listed in Attribute Name. A logical attribute name appears within vertical bars (| |), just as it is defined in the Identity Manager Management Console. For example, a Date logical attribute appears as |Date|.

Example: Assigning a Logical Attribute to a Field

Suppose a newsletter publisher makes portions of its newsletter available online. Users register for the site through the default Identity Manager self-registration screen. The only modification to the default screen is an additional field that asks users whether they currently subscribe to the printed newsletter. A logical attribute handler places subscribers and non-subscribers in different organizations when they self-register for the online version. The value supplied to the field configured with the logical attribute is not written to the data store.

The following steps show how to add the new field and configure it with the custom logical attribute |subscriber|. (Note that the logical attribute would first be defined in the Identity Manager Management Console. It is not one of the predefined logical attributes shipped with Identity Manager.)

1. Click Admin Roles > Modify Admin Task in the Welcome screen of the Identity Manager User Console.
2. Click Search to display a list of all admin tasks.

3. Select the Self Registration radio button, then click Select.
4. Click the Tabs tab in the Modify Admin Task Self Registration screen.
5. Click the right-pointing arrow next to the Profile tab in the list of tabs.
6. Click the ellipsis button (...) to the right of the Screen field in Configure Profile, . This displays a list of profile screens that you can edit.
7. Make sure that the Self Registration Profile radio button is selected in Select Screen Definition, then click Edit.
8. Add a new field to the Self Registration screen in Configure Standard Profile Screen by checking the box to the left of Password, then clicking the top Add button.
9. Click the right-pointing arrow to the right of the newly added field. This displays a list of field properties you can define for the new field.
10. Select the name of the logical attribute to assign to the new field—in this case, |subscriber|, in the Attribute Name field within the Field properties box, .

Note: For |subscriber| to be listed in Attribute Name, it must first be defined in the Management Console.

The Field properties box reappears, containing all the properties that apply to the |subscriber| logical attribute.

11. Finish defining field properties and click Apply.

Note: For more information about adding a field to a task screen and configuring the field with an attribute, see the *CA Identity Manager Administration Guide*.

Use Case Example: Formatting a Date Attribute

This section presents an overview of logical attribute design-time and run time activities. The use case involves a user account expiration date.

Design Time Activities

Problem statement

Need to display an account expiration date in a user-friendly format based on separate month, day, and year fields. But the date is stored in the database as a single physical attribute value in the format mm.dd.yyyy.

Solution

Create a logical attribute scheme that associates the physical date attribute with three logical attributes that are displayed on the screen in separate Month, Day, and Year fields.

Note: Although the date in this use case is an expiration date, the custom logical attribute handler is a general date handler that can process any type of date.

To create a logical attribute scheme for handling expiration dates and other kinds of dates:

1. Register the new logical attribute handler in the Identity Manager Management Console:
2. Create the logical attribute handler `DateFormatAdapter` by extending the class `LogicalAttributeAdapter`. The custom handler includes the implementation of the following methods:
 - `toLogical()`. Converts the date physical attribute value to three separate logical attribute values.
 - `toPhysical()`. Converts the three logical attribute values back to a single physical attribute value.

To obtain the names of the logical and physical attributes, you pass the values of the attribute Name field as defined in the Identity Manager Management Console into the `getLogicalAttributeName()` or `getPhysicalAttributeName()` method. These methods are in the `LogicalAttributeContext` object, which is passed into every call to `DateFormatAdapter`.

In this example, the attribute names you pass in are Month, Day, Year, and Date:

```
private final String LOGICAL_ATTRIBUTE_MONTH = "Month";
private final String LOGICAL_ATTRIBUTE_DAY = "Day";
private final String LOGICAL_ATTRIBUTE_YEAR = "Year";
private final String PHYSICAL_ATTRIBUTE_DATE = "Date";
. . .

public void initialize(LogicalAttributeContext attrContext)
                        throws Exception {
    String _month = attrContext.getLogicaAttributeName(
        LOGICAL_ATTRIBUTE_MONTH);
    String _day = attrContext.getLogicaAttributeName(
        LOGICAL_ATTRIBUTE_DAY);
    String _year = attrContext.getLogicaAttributeName(
        LOGICAL_ATTRIBUTE_YEAR);
    String _date = attrContext.getPhysicalAttributeName(
        PHYSICAL_ATTRIBUTE_DATE);
    . . .
}
```

The handler uses the returned logical and physical attribute names when setting and retrieving attribute values.

3. Deploy the compiled logical attribute handler `DateFormatAdapter`.
4. Restart the application server.
5. When creating the associated task screen in the Identity Manager User Console, assign the three logical attribute names to the Month, Day, and Year fields you are placing on the screen.

It is not necessary to assign the corresponding physical attribute to a field on the task screen.

Run Time Activities

The following activities occur when the end user executes a task on the task screen:

1. Before displaying the task screen, Identity Manager retrieves the value of the Date physical attributes (mm.dd.yyyy) from the database, and then calls toLogical() in the handler.

The toLogical() method's custom code converts the physical attribute value to the three logical attribute values. Identity Manager then displays the logical attribute values in the associated Month, Day, and Year fields.

2. While working in the task screen, the end user decides to an account, which is about to expire. To do so, the user changes the expiration date, and then submits the task screen.

Identity Manager calls the toPhysical() method. The method's custom code converts the logical attribute values in the Month, Day, and Year fields into the associated Date physical attribute value. Identity Manager then writes the Date value in the data store.

Screen-Defined Logical Attributes

Screen-defined logical attributes are Profile tab screen fields on the User Console without any direct associated physical attributes. Screen-defined logical attributes are local to the admin task screen in which they are created, and are implemented entirely as screen fields. Screen-defined logical attributes enable Identity Manager administrators to define and collect information locally, and not persist it to a managed object.

Screen-defined logical attributes are implemented as JavaScript methods associated with specific screen fields. The JavaScript is defined in the screen field properties when editing a Profile tab. Profile tabs can be edited in a Configure Standard Profile Screens, which are accessible from Profile tab configuration screens.

Screen-defined logical attributes are configured in the same manner as other attributes except that you can also define the attribute name. Screen-defined logical attributes are identified by the vertical bar characters (|) which enclose the attribute name, for example:

|City State Zip|

You can define any of the following control styles as screen-defined logical attributes:

- Check Box Multi-Select
- Dropdown
- Dropdown Combo
- Multi-Select
- Option Selector
- Option Selector Combo
- Radio Button Single-Select
- Single-Select

Display the Configure Standard Profile Screen

The Configure Standard Profile Screen allows you to create or modify screen-defined logical attributes.

To display the Configure Standard Profile Screen

1. In the Identity Manager User Console, select Roles and Tasks, Admin Tasks, Modify Admin Task.
A Select Admin Task screen appears.
2. Search for the task you want to modify, and click Select.
A Modify Admin Task screen appears.
3. On the Tabs tab, click the right-pointing arrow next to the Profile tab.
The Configure Profile screen appears.
4. Click the ellipsis button (...) to the right of the Screen field.
The Select Screen Definition list appears.
5. Click the radio button next to the tab you want to modify, and click Edit.
The Configure Standard Profile Screen appears.
6. To create or modify a screen-defined logical attribute, do one of the following:
 - Click the right-pointing arrow next to the field you want to modify.
 - Click an Add button to define a new field.

Note: For more information, see the User Console online help for the Configure Standard Profile Screen.

Initializing Screen-Defined Logical Attributes

Screen-defined logical attributes are not persisted, and any initial value must be derived from other information such as a constant, global information, a default value, a value of another attribute, a JavaScript function, or left as blank.

To initialize a value, an optional JavaScript function can be added to the "Initialization JavaScript" text box of the of the Configure Standard Profile Screen. A FieldContext object is passed to the function when a field needs to be initialized.

The object can be used to set field values, as shown in the following code example:

```
function init(FieldContext){
    var city = FieldContext.getFieldValue("City" );
    var state = FieldContext.getFieldValue("State" );
    var zip = FieldContext.getFieldValue("Zip" );
    FieldContext.setValue( city + " " + state + " " + zip );
}
```

Updating Screen-Defined Logical Attributes

Screen-defined logical attributes can be updated. However since they are not persisted directly, updated values can be saved only by changing other attributes in the subject, or by some other means. The method for doing this is to add JavaScript that validates the data and updates the fields.

To validate, an optional JavaScript function can be added to the "Validation JavaScript" text box of the of the Configure Standard Profile Screen. A FieldContext object is passed to the function when a field needs to be validated.

The object can be used to validate field values, as shown in the following code example:

```
function validate(FieldContext, attributeValue, changedValue, errorMessage){
  if (attributeValue == "" ){
    return true;
  }
  var split = attributeValue.split(" ");
  if ( split.length < 3 ){
    errorMessage.reference="City State Zip expected";
    return false;
  }
  FieldContext.setFieldValue( "City", split[0] );
  FieldContext.setFieldValue( "State", split[1] );
  FieldContext.setFieldValue( "Zip", split[2] );
  return true;
}
```


Chapter 3: Business Logic Task Handler API

This section contains the following topics:

[API Overview](#) (see page 63)

[API Components](#) (see page 67)

[Access to Managed Objects](#) (see page 70)

[Calling Sequence](#) (see page 72)

[Custom Messages](#) (see page 73)

[JavaScript Handler Implementation](#) (see page 74)

[Configuring Business Logic Task Handlers](#) (see page 77)

API Overview

The Business Logic Task Handler API lets you perform custom business logic during data validation operations for the current task—for example:

- Validating customer-specific task screen fields (for example, the value of an Employee ID field must exist in the master Human Resources database).
- Enforcing custom business rules (for example, an administrator cannot be allowed to manage more than five groups).

Business logic task handlers can be implemented in Java or server-side JavaScript.

API Summary

Operation

Task-Level Validation

Called by

Identity Manager

When called

Depending on what methods are implemented, the custom handler can be called at the following points during task execution:

- When a task session is created (to initialize or process task data)
- After selecting a subject from the search result, but before presenting data to a user

- After a user submits a task, but before any security checks
- After security checks are done, but before the asynchronous phase

Operates on

- Identity Manager objects associated with any task screen in the Identity Manager environment. Performed by global handlers. (Java implementations only.)
- Identity Manager objects associated with the particular task screen where the handler is specified. Performed by task-specific handlers. (Java or JavaScript implementations.)

Object update

In these ways:

- In the task session, through the get... methods in BLTHContext. Because Identity Manager generates events for these updates before writing the data to the data store, the updates can be subject to workflow approval processes, auditing, and security checks.
- Directly in the data store, through the provider objects. No events are generated for these updates, so any workflow approval, auditing, and security checks are bypassed.

Can validate?

User input on a task screen against task screen field values, information in Identity Manager objects, and information in third-party data sources.

Note: You can also perform attribute validation using JavaScript and regular expressions. For information, see the *CA Identity Manager Configuration Guide*.

Task Session

A *task session* is an instance of an executing task. Information is stored in a task session as name/value pairs.

During task execution, business logic task handlers have access to the information in the current task session. Task session information is available through the BLTHContext object passed to business logic task handlers.

A task session includes the following:

- Task context—that is, the task being performed, the administrator who is performing the task, and the Identity Manager managed object affected by the task. This managed object is called the *subject* of the task. For example, a User object is the subject of a Create User task.
- An object (BLTHContext) for retrieving the task's subject and its relationships (such as roles and groups).
- Other task session information, such as task creation time and the locale of the administrator who is performing the task.

More Information:

[Access to Objects in a Task](#) (see page 121)

Task Sessions and Managed Objects

Managed objects are available for retrieval from the task session after a search operation is performed for the object—for example, when a user performs a search on a task screen.

If you access a managed object through a BLTHContext get... method, Identity Manager has already performed the search.

Task Session Management

Identity Manager manages the task session.

A task session begins when an administrator selects a task to perform in the Identity Manager User Console. Business logic task handlers are processed when the administrator clicks Submit on the task screen for the selected task.

Because business logic task handlers are processed before Identity Manager creates events for the task, errors can be immediately presented to the administrator performing the task.

Scope of Business Logic Task Handlers

A business logic task handler can have task-specific or global scope:

- *Task-specific* business logic task handlers apply to a particular task or set of tasks. The code is specified directly in the task screen (either inline or by reference) during task screen configuration.

Task-specific business logic task handlers can be written in Java or JavaScript.

- *Global* business logic task handlers can apply to any task in the Identity Manager environment. They are useful for corporate policies that need to be enforced across many or all tasks.

Global business logic task handlers can only be written in Java. The compiled Java class is deployed in the environment's custom directory.

More Information:

[Task-Specific Business Logic Task Handlers](#) (see page 78)

[Deploying Run Time Files](#) (see page 146)

Use Case Examples

Examples of how the Business Logic Task Handler API can be used:

- Check whether a group membership limit is reached

When an administrator attempts to add a member to a group by submitting a Modify Group task for execution, access the Group managed object to determine the number of current members. If the group has the maximum number of members, throw an exception.

- Verify the job title of a new administrator

When a user submits a Create Administrator task for execution, access a third-party database to determine the job title of the administrator being created. If this particular person does not have the job title Supervisor (the only job title authorized to be an administrator), throw an exception.

- Check for duplicate objects

Identity Manager is shipped with a predefined business logic task handler that checks for possible duplication of objects prior to object creation. For example, the name of a newly created role must be unique. When a role is about to be created, the predefined business logic task handler validates whether the name of the new role already exists. If the name does exist, the handler throws an exception indicating that the role name must be unique, and the administrator is given the opportunity to change the name and resubmit the task.

- **Disable a user account after the maximum failed login attempts**

After a user fails to log in three consecutive times, the account is immediately disabled in the data store. Because the account is updated directly in the data store rather than in the task session, Identity Manager does not generate an event for this action, and any workflow approval process, auditing, and security checks that may be associated with this action are bypassed. After the final login attempt, a custom message advises the user that the account is disabled.
- **Assign a user to a group during self-registration**

A user who self-registers with the job title Benefits Administrator is assigned to the group HR. This update is made in the task session, through a call to `assignResources()` in `RelationshipTabHandler`. This update triggers the event `AddToGroupEvent`, allowing the group assignment to be subject to auditing and workflow approval.
- **Assign “manager” attribute for a new user**

An administrator wants to set a value of “manager” before the profile is displayed on the screen. A business logic task handler could be written that would implement the `handleSetSubject()` method for initialization. This method would assign the administrator’s name to the attribute for the manager’s name. If the name of the administrator is not available, an exception is thrown.

API Components

The following components are in the package `com.netegrity.imapi`. The list includes the core object `IMPersistentContext` and a description of how it is used with this API:

BLTHAdapter

The base class that all business logic task handlers extend. Contains base implementation methods for startup and shutdown operations. Also, `BLTHAdapter` determines whether the handler applies to the task currently being performed, based on the `BLTHContext` information passed into it. If the handler does apply to the current task, the handler performs the validation.

Implements `LifeCycle` and `Logger`.

BLTHContext

Interface that provides methods for retrieving the run time instances of managed objects in the current task session—for example:

- The subject of the current task (the managed object that is directly affected by the current task)
- Any relationship objects (such as roles and groups) that the subject might be assigned to

This interface also provides access to IMContext information about the current task, and to logging methods and other Identity Manager services, including managed object retrieval from the data store.

When a user submits a task for execution on an Identity Manager task screen, BLTHContext information for the task is passed to the methods in the BLTHAdapter object.

Extends IMPersistentContext.

IMPersistentContext

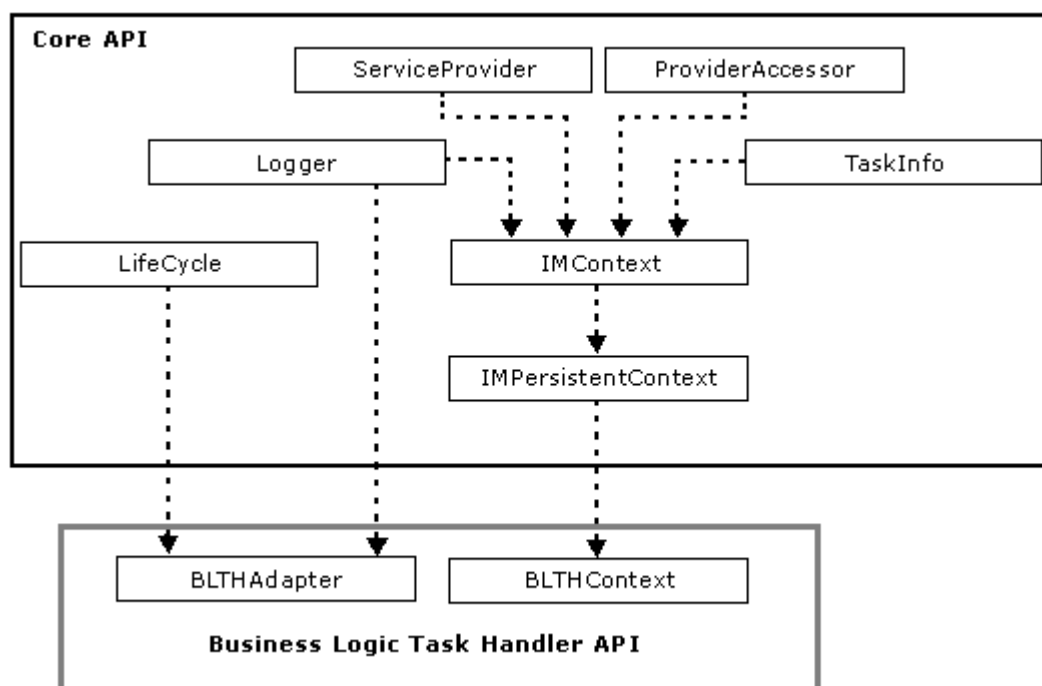
Allows user-defined data to be passed between business logic task handlers executed for a given task session, and between business logic task handlers and other custom objects in the same task session. The persisted data is abandoned after the task session ends.

This interface is part of the core API functional model.

Extends IMContext.

Business Logic Task Handler API Model

The components of the Business Logic Task Handler API are part of the Identity Manager API functional model:



Sample Business Logic Task Handlers

Sample implementations of business logic task handlers are installed in the samples\BLTH subdirectory of your Identity Manager installation—for example: C:\Program Files\CA\Identity Manager\samples\BLTH

Java and JavaScript samples are included.

Note: For information about using a sample, see the readme.txt file that accompanies each sample.

Access to Managed Objects

A business logic task handler can access a managed object in these ways:

- Using the Identity Manager provider objects to access a managed object in the data store.

Any changes made to a managed object through the providers are immediately committed to the data store. No Identity Manager events are generated, and no workflow approvals, auditing, or security checks are performed.

- Using the Identity Manager BLTHContext get... methods to access a run time instance of a managed object in the task session.

The information you can access in a task session includes the subject of the task (such as a User object in a Create User task) and the subject's relationships (such as the roles and groups that a user is assigned to). You can also access any objects that were modified during the task (such as user objects that were disabled or deleted).

Any changes made to a managed object through BLTHContext get... methods are made to the run time instance of the managed object in the task session. Identity Manager generates events for the task, and workflow approvals, auditing, and security checks can be performed.

The changes are not committed to the data store until all events associated with the task have been executed.

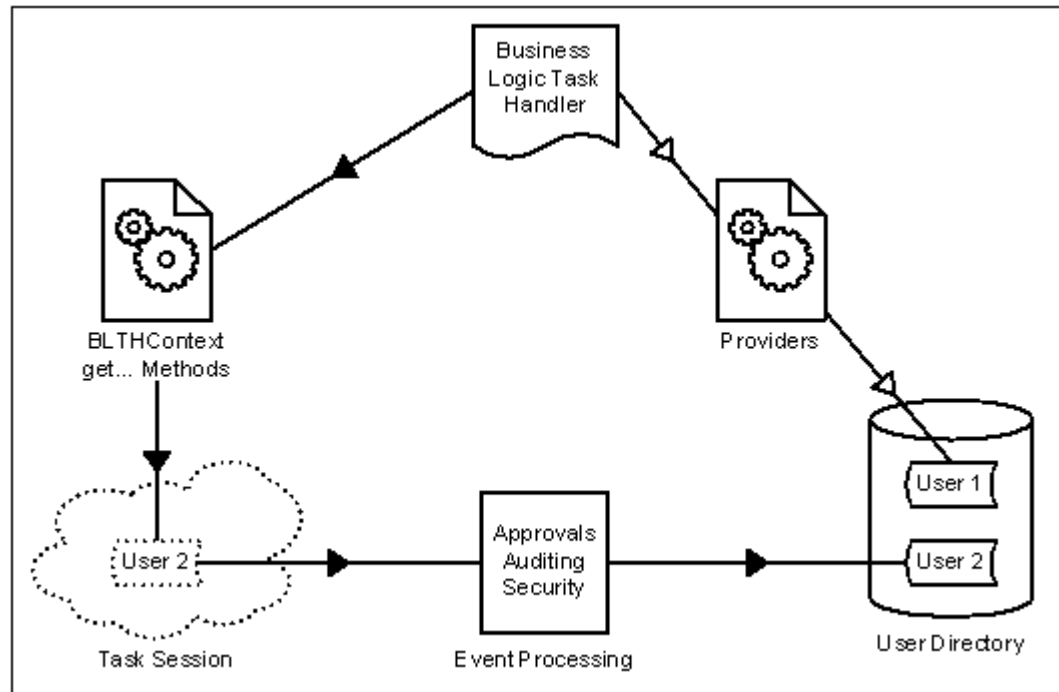
More Information:

[Access to Objects in the Data Store](#) (see page 117)

[Access to Objects in a Task](#) (see page 121)

Managed Object Process Diagram

In this diagram, a business logic task handler modifies User 1 directly in the data store through the providers, and it also modifies a run time instance of User 2 in the task session. When all event processing and associated operations for the User 2 updates are complete, Identity Manager commits User 2 to the data store.



Calling Sequence

Identity Manager calls business logic task handlers after all logical attribute handler processing is complete, so that business logic task handlers have access to the logical attribute values and the physical attribute values set by the logical attribute handlers.

Business logic task handlers are invoked in this sequence:

1. An Identity Manager admin task is submitted on a task screen.
2. Identity Manager creates a BLTHContext object for the task, and passes this object to each business logic task handler it invokes.
3. If any task-specific handlers are defined for the task, Identity Manager invokes the first handler referenced on the task screen.
4. After all task-specific handlers are executed, Identity Manager invokes any global task handlers defined in the Identity Manager Management Console for the environment, beginning with the first handler defined on the Business Logic Task Handlers screen.

If a business logic task handler throws an exception (or if a JavaScript handler returns false), the task screen is redisplayed with one or more exception messages, giving the administrator a chance to address any errors and resubmit the admin task.

To create a custom business logic task handler, you write a class that extends the base class BLTHAdapter. You can implement your adapter so that Identity Manager can execute the business logic at various points during the synchronous phase of task execution. You can implement any one or all of the following methods in your custom adapter:

- To execute business logic at the beginning of the task, implement `handleStart()`.
- To execute business logic after selecting the subject from the search result, implement `handleSetSubject()`.
- To execute business logic after the user has submitted the task, but before the security check, implement `handleValidation()`.
- To execute business logic after a security check and the events are ready to be posted, implement `handleSubmission()`.

Example: Calling a Business Logic Task Handler

Identity Manager invokes any method implemented for a registered handler at the appropriate time. For example, this code fragment shows an implementation of the `handleSetSubject()` method to extract an administrator name before presenting the data to the user:

```
import com.netegrity.imapi.BLTHAdapter;
import com.netegrity.imapi.BLTHContext;
import com.netegrity.ims.exception.IMSEException;

public class BLTHSetManager extends BLTHAdapter {
    public void handleSetSubject(BLTHContext blthContext) throws Exception {
        // get current administrator name
        String managerUniqueName = blthContext.getAdminUniqueName();
        if (managerUniqueName == null) {
            // this message will be presented on the screen
            IMSEException imsEx = new IMSEException();
            imsEx.addUserMessage("Failed to get administrator unique name");
            throw imsEx;
        }
    }
}
```

Custom Messages

After a user submits a task, Identity Manager can display the following kinds of use-defined messages:

- Informational messages
- Exception messages

Informational Messages

A business logic task handler can define a custom informational message, such as an acknowledgement, to display to the user after the user successfully submits a task. For example, after a user submits a request to open an account, the following task-specific message might be displayed:

Your request for an account with XYZBank is being processed. Please go to www.xyzbank.com/welcome.html to learn about the benefits of being an XYZBank customer.

To display a custom informational message for the task, specify the message in an `addMessageObject()` method inherited by `BLTHContext`.

Exception Messages

If validation errors occur after a user submits a task screen, the task screen is redisplayed with the appropriate exception messages. This gives the user the opportunity to correct the errors and resubmit the task screen.

To enable Identity Manager to display custom exception messages on a task screen, add the messages to an `IMSEException` object.

Identity Manager supports localized exception messages, which you store in a family of resource bundles. Each resource bundle in the family contains the same messages, but in a different language. You specify the current user's locale and the base name of the resource bundles, and Identity Manager selects the appropriate resource bundle from which to retrieve exception messages.

Note: For detailed information see the `IMSEException` class in the Javadoc reference.

More Information:

[IMSEException](#) (see page 134)

Custom Message Example

The following code example shows how to display a custom informational message using the `addMessageObject` method.

The code example uses the methods `ProcessStep` and `ErrorLevel`. The field `ErrorLevel.SUCCESS` instructs Identity Manager to display the message on the screen:

Example:

```
IMSEException imsx = new IMSEException();
imsx.addUserMessage("My custom message");
context.addMessageObject (imsx,"MyBLTH", ProcessStep.DATAVALIDATE,
ErrorLevel.SUCCESS);
```

JavaScript Handler Implementation

You can use JavaScript to write task-specific business logic task handlers.

Task-specific handlers are defined on a task screen, either inline or by reference.

For BlthContext methods and objects, the JavaScript business logic task handlers do not have to explicitly import any Identity Manager packages. JavaScript handlers have automatic access to the BlthContext methods and objects, the same as Java handlers.

However, when referencing imsapi classes without going through the context object, you must fully qualify the reference to the class. To do this, you can define a variable as a shortcut to the imsapi package, and then use this in all other class references where necessary.

JavaScript Method Reference

A JavaScript handler implements the same methods as the Java BLTHAdapter class:

```
handleStart(BLTHContext ctx)
handleSetSubject(BLTHContext ctx)
handleValidation(BLTHContext ctx)
handleSubmission(BLTHContext ctx)
```

Example:

```
// This sample illustrates BLTH implemented in java script
// It verifies that a tag specified in the task is unique in
// the environment function

handleValidation(blthContext, errorMsg) {
    var taskTag = null;
    var currentTask = blthContext.getAdminTask();
    if(currentTask != null) {
        taskTag = blthContext.getAdminTask().getTaskTag();
    }

    if(taskTag == null) {
        errorMsg.reference = "Failed to get the task tag";
        return false;
    }
    try {
        task = blthContext.getAdminTaskProvider().findByTag(taskTag);
    } catch(exception) {
        // there is no task with this tag
        return true;
    }
}
```

```
    if (!currentTask.equals(task)) {
        errorMsg.reference = "Tag must be unique: " + taskTag;
        return false;
    } else {
        // the same task
    }

    return true;
}
```

IMSAPI Class References

The following code samples demonstrate explicitly how to reference imsapi classes:

Sample 1:

```
function handleValidation(BlthContext, errorMessage) {
    var imsapi = Packages.com.netegrity.llsdk6.imsapi
    var expr = new imsapi.search.SearchExpression("uid",
        imsapi.type.OperatorType.EQUALS, "bhanu")
    return true
}
```

Sample 2:

```
function handleValidation(BlthContext, errorMessage) {
    var searchPackage = Packages.com.netegrity.llsdk6.imsapi.search
    var typePackage = Packages.com.netegrity.llsdk6.imsapi.type
    var expr = new searchPackage.SearchExpression("uid",
        typePackage.OperatorType.EQUALS, "bhanu")
    return true
}
```

Sample 3:

```
function handleValidation(BlthContext, errorMessage) {
    var expr = new
    Packages.com.netegrity.llsdk6.imsapi.search.SearchExpression("uid",
        Packages.com.netegrity.llsdk6.imsapi.type.OperatorType.EQUALS, "bhanu")
    return true
}
```

Configuring Business Logic Task Handlers

Business logic task handlers are configured in the following locations:

- Global business logic task handlers are configured in the Identity Manager Management Console. These task handlers can be used with any of the tasks in the Identity Manager environment.
- Task-specific business logic task handlers are configured in the particular task screen where the handler will be used.

Global Business Logic Task Handlers

You define a global business logic task handler's fully qualified class name and other properties in the Identity Manager Management Console.

To configure a global business logic task handler:

1. Click Identity Manager environments in the Identity Manager Management console.
2. Click the name of the environment where the business logic task handler will be used.
3. Click Advanced Settings.
4. Click Business Logic Task Handlers.

The Business Logic Task Handlers screen contains a list of existing global business logic task handlers. The list includes predefined handlers shipped with Identity Manager and any custom handlers defined at your site. Identity Manager executes the handlers in the order in which they appear in this list.

Global business logic task handlers can be implemented only in Java.

From the Business Logic Task Handlers screen, you can do the following:

- Register a new business logic task handler with Identity Manager. To begin this operation, click New.
- Modify an existing handler configuration. To begin this operation, click the handler name.
- Delete a handler.
- Change the order of handler execution.

For detailed instructions about performing these operations, press the help link on the Business Logic Task Handlers screen.

More Information:

[Compiling and Deploying](#) (see page 145)

Task-Specific Business Logic Task Handlers

You define a task-specific business logic task handler's fully qualified class name and other properties in the Identity Manager User Console.

To define a task-specific business logic task handler:

1. Navigate in the User Console to the configuration screen for the task you are associating with the business logic task handler.
2. Click Business Logic Task Handlers.
3. The Business Logic Task Handlers screen appears. This screen lists any existing business logic task handlers assigned to the task. Identity Manager executes the handlers in the order in which they appear in the list.
4. Click Add.

The Business Logic Task Handler Detail screen appears.

For detailed instructions about configuring task-specific business logic task handlers, click the Help link on the Business Logic Task Handler Detail screen.

Chapter 4: Event Listener API

This section contains the following topics:

[Tasks and Events](#) (see page 79)

[Event Listener Overview](#) (see page 80)

[Event Listener Execution](#) (see page 84)

[Access to Managed Objects](#) (see page 88)

[Event Generation](#) (see page 89)

[Event Listener Configuration](#) (see page 89)

Tasks and Events

An Identity Manager task is made up of one or more events. When Identity Manager executes a task, it breaks the task down into its component events. Each event represents a specific action to be performed during the execution of the task. For example, a Create User task might consist of the following:

- A CreateUserEvent, to add the new user to an organization in the directory
- Possibly, one or more AddToGroupEvent events, to add the new user to a group
- Possibly, one or more AssignAccessRoleEvent events, to assign the new user to a role
- Possibly, other secondary events generated for a Create User task.

Breaking down a task into its component events allows access to the actions that occur within the task—for example:

- A workflow process is associated with an individual event rather than the overall task. Consequently, if a workflow process that is mapped to a secondary event (such as AddToGroupEvent or AssignAccessRoleEvent) fails, the primary event (such as CreateUserEvent) and other secondary events can still occur successfully.
- Different sets of participants can be assigned as approvers for different events within the same task. For example, in a Create User task, one set of participants can be defined as approvers for adding the new user to a group (AddToGroupEvent) and another set of participants can be defined as approvers for adding the user to a role (AssignAdminRoleEvent).
- Event listeners perform custom business logic when a particular event within a task occurs—more specifically, during a particular state of an event.

Event Listener Overview

An event listener "listens for" a specific event or a group of events. When the event occurs, the event listener performs custom business logic that is appropriate for the event and the current event state.

Event listeners are invoked during the following event states:

Pre-approval

Occurs before the workflow process associated with the event begins. If no workflow process is associated with the event, this state occurs before the Approved state. An event that is not associated with a workflow process is approved automatically.

Approved

Occurs after a workflow-controlled event is approved, but before the event is executed. This state is generated automatically for an event that is not associated with a workflow process.

Rejected

Occurs after the event is rejected. Any updates made to a managed object during this state are not written to the data store.

Post-execution

Occurs after execution of the event is completed.

More Information:

[Event Listener State Diagram](#) (see page 86)

Event Listener Operations

This summarizes the Event Listener API:

Perform Pre-Approval Attribute Updates

Called by Identity Manager before a workflow process begins, or before the automatic approval of an event that is not associated with a workflow process

Operates on events

Method called: `before()`

Object updates:

- The run time instance of the managed object in an event.
- Data store objects, through the providers. No events are generated for these updates, so any workflow approval, auditing, and security checks are bypassed.

Cannot validate

Perform Pre-Execution Attribute Updates

Called by Identity Manager after a workflow-controlled event is approved or rejected. If the event is not associated with a workflow process, approval is automatic

Operates on events

Methods called: `approved()` `rejected()`

Object updates:

- The run time instance of the managed object in an event.
- Data store objects, through the providers. No events are generated for these updates, so any workflow approval, auditing, and security checks are bypassed.

Cannot validate

Perform Post-Execution Operations

Called by Identity Manager after the event is executed

Operates on events

Method called: `after()`

Object update only through the providers

Cannot validate

Use Case Examples

Event listeners can listen for a specific event, a type of event (such as user events or organization events), or all events. The following are examples of how you can use the Event Listener API:

Add a user to Microsoft Exchange (specific event listener)

When a user is successfully added to the admin role Exchange Users, Identity Manager calls the method `after()` in an event listener associated with the event `AssignAdminRoleEvent`. The implementation of the `after()` method uses an Exchange API to add the user to the Exchange system. The information required to access the Exchange server (`SERVER_NAME`, `ADMIN_ID`, `ADMIN_PASSWORD`, and so on) is included in the user-defined properties of the Identity Manager environment. Identity Manager environment properties are passed to the `init()` method of the `EventListenerAdapter`.

Create a rule for group assignments (specific event listener)

When a user is created, Identity Manager calls the `after()` method in an event listener associated with the event `CreateUserEvent`. The implementation of the `after()` method evaluates user attributes such as Job Title, Security Level, and Department to assign the user to one or more groups. For each group assignment, the implementation generates the secondary event `AddToGroupEvent`. This secondary event can be subject to workflow approvals and auditing.

Update an RDBMS with user data (class-level event listener)

Suppose that user information such as user ID, user name, email address, and group membership is stored in an RDBMS. Whenever any user-level event occurs (`CreateUserEvent`, `ModifyUserEvent`, and so on), the user's record in the RDBMS needs to be updated. After the event is executed, Identity Manager calls the method `after()` in an event listener associated with user events. The implementation of this method retrieves the RDBMS connectivity parameters from the Identity Manager environment properties passed into the `init()` method, and then updates the user record in the RDBMS.

Post events to a JMS queue (all event listener)

After any event is successfully executed, Identity Manager calls the method `after()` in an event listener associated with all events. This event listener writes all events to a specific Java Message Service (JMS) queue. The Java Naming and Directory Interface (JNDI) name for the queue and the information in the `jndi.properties` file should be configurable through the properties of the Identity Manager environment. The `after()` method implementation uses the JNDI information to post a simple object message to the queue.

API Components

The Event Listener API contains the following components.

The list includes the core objects `EventContext` and `IMPersistentContext`, and a description of how they are used with this API:

EventListenerAdapter

The base class that all event listeners extend. Contains base implementation methods for startup and shutdown operations. Also provides entry points for the various event states. Identity Manager calls a different `EventListenerAdapter` method for each event state and passes `EventContext` to the method.

Implements `LifeCycle` and `Logger`.

EventContext

Interface that provides read/write access to managed objects in Identity Manager events, and lets you generate secondary events.

This interface also provides access to `IMContext` information about the current task, and to logging methods and other Identity Manager services, including managed object retrieval from the data store.

Extends `IMPersistentContext`.

`EventContext` is part of the core API functional model. However, it serves as the `...Context` module for the Event Listener API.

IMPersistentContext

Allows user-defined data to be persisted across different event listeners within a given event. Also allows data to be persisted between an event listener and other custom objects in the same task session. The persisted data is not written to the data store.

This interface is part of the core API functional model.

Extends `IMContext`.

The above components are in the package `com.netegrity.imapi`.

Note: For detailed information see the Javadoc reference.

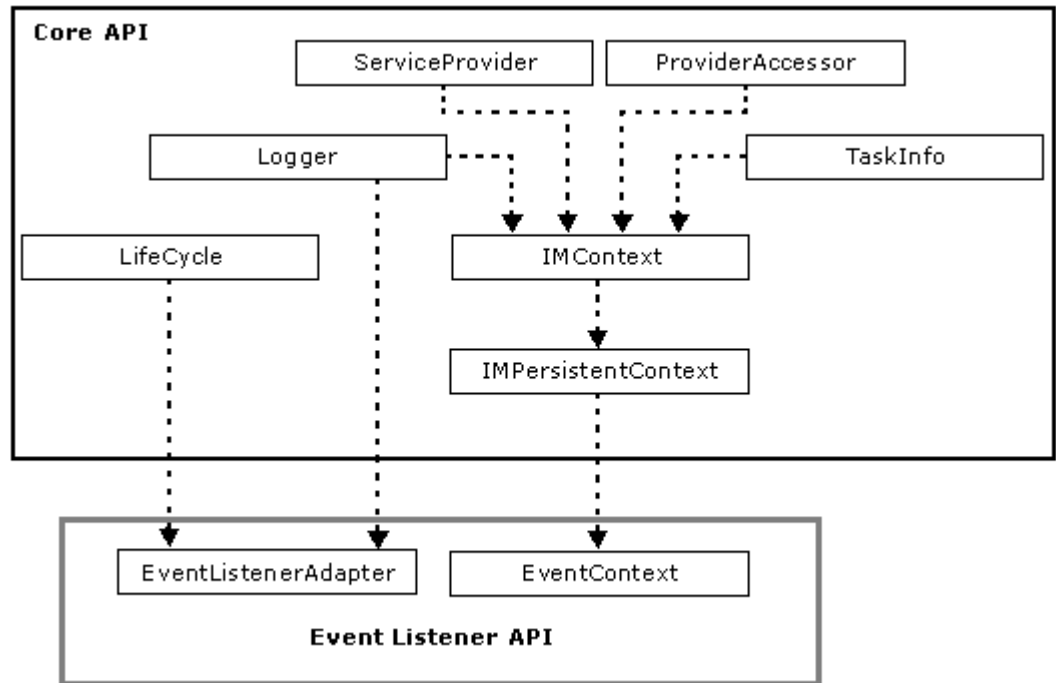
More Information:

[Event Listener States](#) (see page 85)

[Access to Event Objects](#) (see page 124)

Event Listener API Model

The components of the Event Listener API are part of the Identity Manager API functional model:



Event Listener Execution

Identity Manager invokes an event listener when both of the following actions take place:

- An event associated with the event listener occurs
- The event moves into a given event state

For the event listener to "listen for" an event, you must map the event listener to one or more events. You do so in the Event Listener Properties screen of the Identity Manager Management Console.

You can map an event listener to one of the following:

- **A specific event.** Identity Manager invokes the event listener when the specified event occurs.
- **A category of events**, called a listener level. Identity Manager invokes the event listener when any of the events in the listener level occurs.
- **All events.** Identity Manager invokes the event listener when any event occurs.

More Information:

[Event Listener Configuration](#) (see page 89)

Event Listener States

When an event that is mapped to an event listener occurs, Identity Manager invokes the event listener during each of the states in the following table. The states occur in the following order:

Pre-approval state

This state occurs:

- Before a workflow-controlled event is approved or rejected
- Before the automatic approval of an event that is not workflow controlled

Entry Point in EventListenerAdapter: before()

Approved state or Rejected state

- **Approved.** Occurs after a workflow-controlled event is approved. This state also applies to events that are not under workflow control. These events are automatically approved.
- **Rejected.** Occurs after a workflow-controlled event is rejected. This state applies only to events that are under workflow control.

Entry Point in EventListenerAdapter: approved() or rejected()

Post-execution state

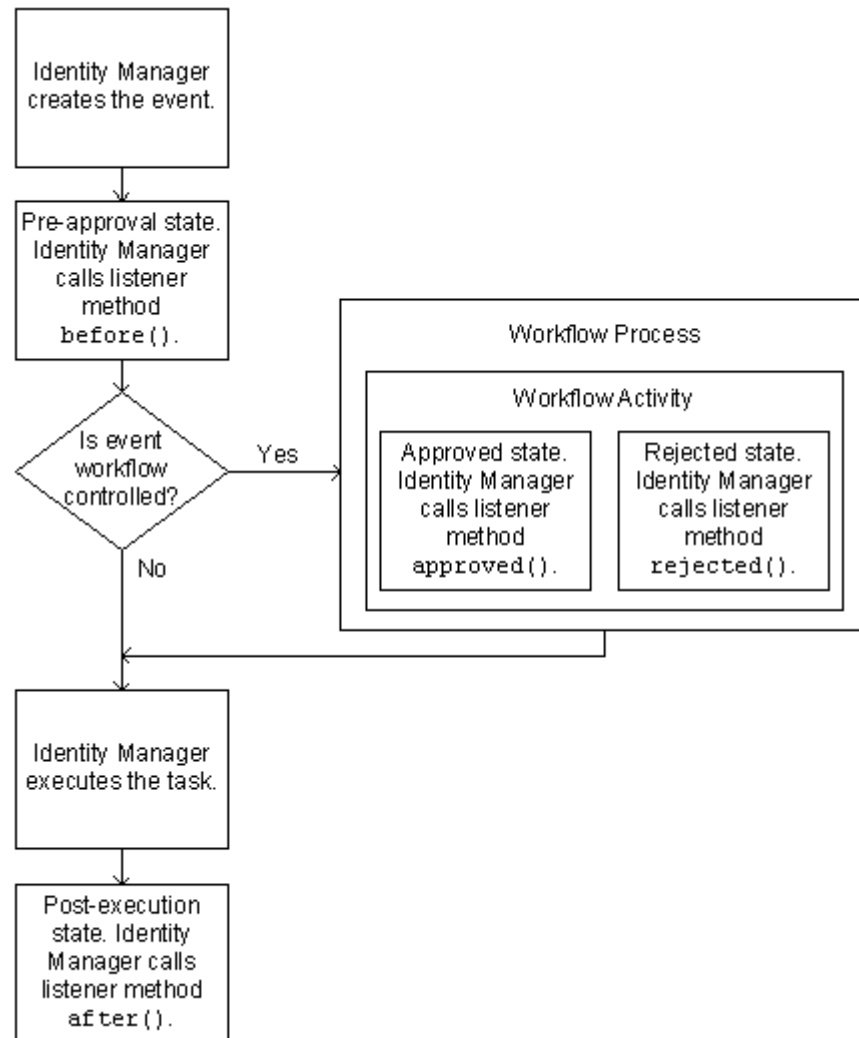
Occurs after the event is executed. Provisioning (automatic, rule-based role and group assignments) are typically made during this state.

Entry Point in EventListenerAdapter: after()

Event Listener State Diagram

The following state diagram shows the various states of event processing. If an event listener is associated with the current event, Identity Manager calls the event listener during each event state.

Event Listener Execution During Event States



Sample Event Listener

A sample implementation of an event listener is installed in the `samples\EventAdapter` subdirectory of your Identity Manager installation:

`C:\Program Files\CA\Identity Manager\samples\EventAdapter`

For information about using the sample, see the `readme.txt` file that accompanies the sample.

Order of Execution

When an event occurs, Identity Manager evaluates every event listener referenced in the Identity Manager Management Console for the environment where the event occurred, checking whether any event listeners are mapped to the event. Identity Manager invokes all event listeners mapped to the event.

Identity Manager evaluates and invokes event listeners according to the following rules:

- Event listeners are evaluated in the order of their level of specificity, from most specific to most general. That is, event listeners mapped to specific events are evaluated first, then event listeners mapped to class events, and finally event listeners mapped to all events.
- Event listeners within a given level of specificity are evaluated in the order of their position on the Event Listeners screen of the Identity Manager Management Console, with the topmost event listener being evaluated first.
- If an event listener is mapped to the current event, Identity Manager invokes the event listener at the current event state.
- If an event listener completes its execution for a given state and returns `CONTINUE`, Identity Manager continues as follows:
 - By invoking the next listener for the same event state.
 - After all listeners have been processed for the current event state, Identity Manager advances the event to the next sequential state, and evaluates the next listener according to the rules listed above.
- If an event listener returns `BREAK`, no more event listeners are processed for the current event state. Identity Manager advances the event to the next sequential state, and evaluates the next listener according to the above rules.

The CONTINUE and BREAK return codes cause Identity Manager to advance the event to the next sequential state. The event listener cannot specify a particular event state to enter. To move an event to an Approved or Rejected state during workflow processing, use the WorkflowContext object in the Workflow API.

If an event listener adapter throws an exception, Identity Manager does not complete the execution of the event.

More Information:

[Event Listener State Diagram](#) (see page 86)

Access to Managed Objects

An event listener has access to the following managed objects:

Managed objects in an event.

Each Identity Manager event is represented by an event object. Identity Manager generates event objects and makes the objects available for retrieval by event listeners.

An event listener can retrieve an event object through the methods in the EventContext object that Identity Manager passes into the listener.

Once you have an event object, you can retrieve a run time instance of the managed objects contained in the event.

Managed objects in the data store.

Using the Identity Manager providers, an event listener can retrieve any of the managed objects in the data store through a provider.

If an event listener modifies managed object data through a provider, the changes are immediately committed to the data store. No Identity Manager events will be generated for the changes, and no workflow approvals, auditing, or security checks will be performed.

More Information:

[Retrieving Event Objects](#) (see page 125)

[Base Event Interfaces](#) (see page 125)

[Access to Objects in the Data Store](#) (see page 117)

Event Generation

An event listener can generate a secondary event within the current task.

For example, in a Modify User task, suppose an event listener associated with `ModifyUserEvent` adds a user to a group or removes a user from a group. Identity Manager does not generate an event for an operation such as this when the operation is performed by a custom event listener. However, the event listener can generate the appropriate event (in this case, `AddToGroupEvent` or `RemoveFromGroupEvent`) by calling `generateEvent()` in `EventContext`.

The event generation feature lets an event listener perform rule-based role and group assignments that:

- Can be subject to workflow approvals and auditing, just as if Identity Manager had generated the event.
- Can be associated with a different administrator than the one who initiated the original task.

No Identity Manager security checks are performed on events generated by an event listener.

Event Listener Configuration

To allow an event listener to "listen for" a particular event or group of events, you map the event listener's fully qualified class name to the events. When a mapped event occurs, Identity Manager calls the associated event listener.

You define an event listener's fully qualified class name, its mapped events, and other properties in the Identity Manager Management Console.

To set event listener properties:

1. In the Management Console, click Identity Manager Environments.
2. Click the name of the environment where the event listener will be used.
3. Click Advanced Settings.
4. Click Event Listeners.

For instructions about event listener properties, click Help on the Event Listeners screen.

More Information:

[Compiling and Deploying](#) (see page 145)

Chapter 5: Workflow API

This section contains the following topics:

[Workflow API Overview](#) (see page 91)

[Participant Resolver API](#) (see page 97)

Workflow API Overview

The Workflow API provides the following kinds of information to a custom script in a workflow process:

- Information about the currently executing Identity Manager event
- Information about the administrator who submitted the event
- Information within the Identity Manager environment

The workflow script evaluates the information and determines the path of the workflow process accordingly.

Methods in the Workflow API are called from either of the following:

- A custom workflow script—that is, from scripts in automated workflow activities and scripts in conditional transitions.
- A Java object which is then accessed from a workflow script.

For information about writing workflow scripts and about accessing Java objects from a workflow script, see the appropriate workflow documentation.

Note: For information about using WorkPoint workflow, see the *CA Identity Manager Administration Guide*.

Workflow API Summary

Operation

Custom Objects Perform Operations During Workflow Processing.

Called by

A custom script in an automated workflow activity or a workflow conditional transition, or a Java object which is accessed from a workflow script.

When called

When the workflow script is executed.

Operates on

The Identity Manager event mapped to the workflow process, including Identity Manager objects contained in the event.

Object update

The run time instance of the managed object in an event.

Can validate?

No. However, after evaluating the information, the workflow process can move the event to an accepted or rejected state.

Use Case Examples

The following are examples of how the Workflow API can be used:

Determine workflow direction based on a User object attribute

The first activity in a workflow process for a CreateUserEvent is an automated activity with three conditional transitions leading from it. The condition script in each transition retrieves the email attribute from the User object contained in the event.

The next activity that the workflow process will perform is determined by the outcome of the condition scripts, as follows:

- In the transition NETEGRITY.COM SUFFIX, the condition script checks whether the user's email address ends in netegrity.com. If it does, the script returns true, and the transition is performed. This transition leads to a single approval activity where the addition of the user is determined by an approver who is assigned the role HR Manager.

- In the transition SECURITY.COM SUFFIX, the condition script checks whether the user's email address ends in security.com. If it does, the script returns true, and the transition is performed. This transition leads to an intermediate approval activity (by an approver with the role HR Manager) and a subsequent approval activity (by an approver with the role IT Manager).
- In the transition EMAIL INVALID, the condition script checks whether the user's email address ends in something other than netegrity.com or security.com. If this script returns true, the transition is performed, leading directly to an automated activity that rejects the event CreateUserEvent.

Update an Identity Manager object during a workflow process

In the previous use case example, suppose the condition script for the transition NETEGRITY.COM SUFFIX is updated to check whether the email attribute is blank. If it is blank, the script constructs an email address from the user's ID and the default suffix netegrity.com. The script assigns the address to the email attribute of the User object. The script then returns true, and the transition is performed as in the previous example. The updated object is saved when Identity Manager completes the execution of the event.

Validate an object from a workflow process using third-party data

The workflow process for a CreateUserEvent contains an automated activity that retrieves the new user's credit score from a third-party database. First, the activity script calls the Workflow API to retrieve the event and the User object within the event. Then, after the script retrieves the user's credit score, it uses the workflow engine's API to write the credit score to the User Data for the activity, and finally completes the activity. Transition scripts leading from the activity then use the workflow engine's API to retrieve the credit score from the User Data and evaluate it to determine the subsequent course of the workflow process.

API Components

The Workflow API contains the following components. The list includes the core objects `EventContext` and `IMPersistentContext`, and a description of how they are used with this API:

WorkflowContext

Interface providing access to the current event, including the managed object contained in the event. It also allows a workflow script to approve or reject the activity associated with the current event, and to retrieve the activity's participants.

This interface also provides access to `IMContext` information about the current task, and to logging methods and other Identity Manager services, including managed object retrieval from the data store.

Extends `EventContext`.

EventContext

Provides read/write access to the managed objects in Identity Manager events, and lets you generate secondary events. This interface is part of the core API functional model.

Extends `IMPersistentContext`.

IMPersistentContext

Allows user-defined data to be passed between scripts in a given workflow process, and between a workflow script and a custom object (such as an event listener) in the same task session. The persisted data is not written to the data store.

This interface is part of the core API functional model.

Extends `IMContext`.

The above components are in the package `com.netegrity.imapi`.

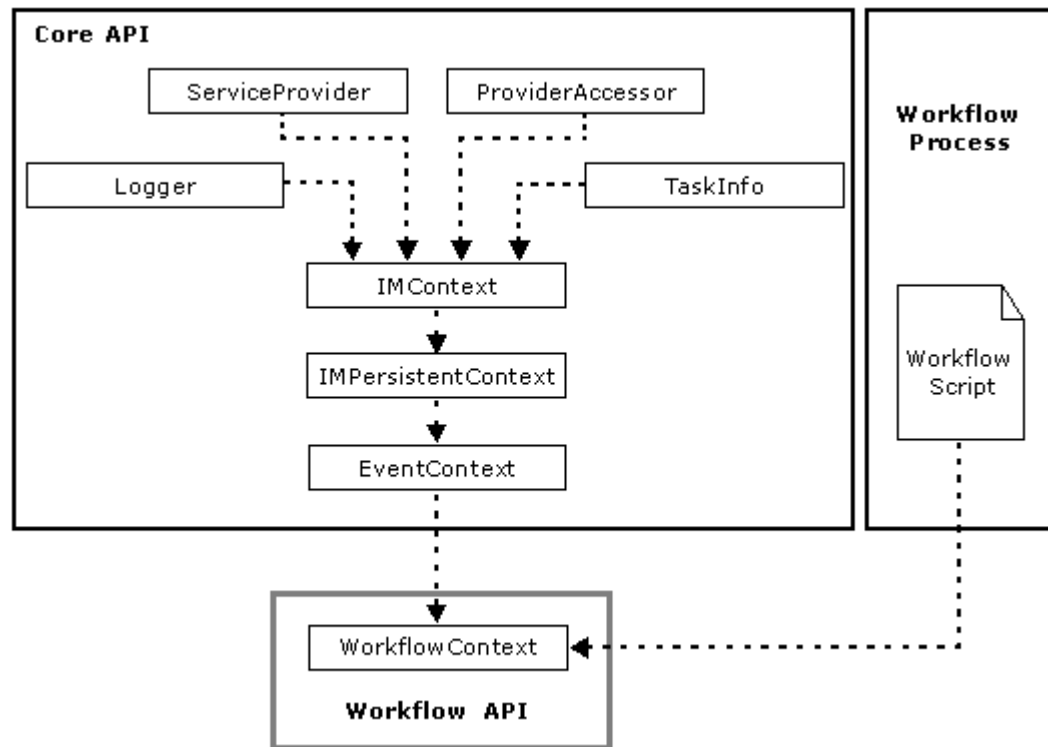
Note: There is no adapter component in the Workflow API because you do not create a custom Java object with this API. You call the methods in the API from a WorkPoint workflow script.

More Information:

[Access to Event Objects](#) (see page 124)

Workflow API Model

The components of the Workflow API are part of the Identity Manager API functional model:



Sample Workflow API Implementation

A sample Java class that calls the Workflow API is installed in the samples\WorkflowContext subdirectory of your Identity Manager installation—for example:

C:\Program Files\CA\Identity Manager\samples\WorkflowContext

Also included are sample workflow scripts that access the Java class.

Note: For information about using the sample Java class and scripts, see the readme.txt file that accompanies the samples.

Integration with Third-Party Applications

The Workflow API lets you integrate third-party applications with information in Identity Manager managed objects and with a workflow process. For example:

- A script in an automated workflow activity can call a third-party application and pass information about Identity Manager objects to the application.

For example, suppose a `CreateUserEvent` event requires approval in a workflow process. If the user is approved, a `User` object will be created in the Identity Manager environment.

In addition, the new user needs to be added to a third-party payroll system. The third-party application requires the user's ID and department number—both of which were defined in the task where `CreateUserEvent` originated.

So, after the `CreateUserEvent` is approved in a manual activity, a transition leads to an automated activity. This activity contains a script that uses the Workflow API to retrieve the required data from Identity Manager and pass the data to the third-party payroll system.

- An automated activity can retrieve information from a third-party application and add the information to the current event.

For example, suppose one of the pieces of information supplied in a `Create User` task is the email address of the new user. An automated activity in the workflow process where the `CreateUserEvent` is approved could check whether email information was provided for the user on the task screen. If it was not, the automated activity's script could construct an email address from information retrieved from the user's profile in a third-party data source, and then update the Identity Manager `User` object with the email address.

When Identity Manager completes the execution of the `CreateUserEvent` event, the email address will be stored along with the other information about the new user.

Participant Resolver API

The Participant Resolver API determines the participants in a workflow activity.

A *participant* is a person who is authorized to perform a workflow activity. In Identity Manager, participants are also called *approvers*, since they must approve or reject the task under workflow control.

Operation

Custom Objects Perform Operations During Workflow Processing.

Called by

Identity Manager.

When called

When a workflow activity begins.

Operates on

The workflow activity that requires the participants.

Object update

Not relevant.

Can validate?

Not relevant.

About Participant Resolvers

A workflow activity must reference the list of participants who are authorized to approve a workflow activity. This reference is made in the activity's properties, which are defined in the user interface for the particular workflow engine you are using.

Rather than hard-code a list of participants into a workflow activity's properties, the participants are referenced by an arbitrary name that is mapped to a *participant resolver*.

There are several types of participant resolvers:

Role type

Participants are the members of a particular role.

Group type

Participants are the members of a particular group.

Filter type

Participants are selected through a search filter.

Custom type

Participants are determined by a custom participant resolver.

Note: For information about other types of resolvers, see the *CA Identity Manager Administration Guide*.

Custom Participant Resolvers

The custom participant resolver is a Java object that determines a workflow activity's participants and returns it to Identity Manager. Identity Manager then passes the list to the workflow engine.

Typically, you will write a custom participant resolver only if none of the standard resolver types can provide the list of participants that an activity requires.

You map a custom participant resolver to a fully qualified participant resolver class in the Identity Manager Management Console.

More Information:

[Participant Resolver Configuration](#) (see page 102)

Use Case Example

The following is an example of how the Participant Resolver API can be used:

Assign a group's administrators as participants

Workflow approval is required whenever a user is added to the group HR. A custom participant resolver assigns the administrators of the group as participants in the workflow activity. When a user is added to HR, the event `AddToGroupEvent` comes under workflow control, and cannot be completed successfully until a designated participant approves the activity.

API Components

The Participant Resolver API contains the following components. The list includes the core object EventROContext and a description of how it is used with this API:

ParticipantResolverAdapter

The base class that all participant resolvers implement. Contains base implementation methods for startup and shutdown operations. Also returns a Vector of workflow activity participants to Identity Manager.

Identity Manager passes ParticipantResolverContext information into the base class. Identity Manager also passes in the name of the workflow approval task, and a hashtable of properties defined in the workflow engine's user interface (for example, in the User Data tab of the WorkPoint Activity Properties dialog). This information includes:

- The attribute APPROVERS_REQUIRED, which specifies whether at least one participant must be found before the activity can be completed.
- Any user-defined data required by your custom participant resolver and that is defined in the User Data tab.

Implements LifeCycle and Logger.

ParticipantResolverContext

Interface containing information about the current task—for example, the task name, the administrator who is performing the task, and the organization where the subject of the task is located.

The ParticipantResolverContext interface also provides access to logging methods and other Identity Manager services, including managed object retrieval from the data store.

Identity Manager passes ParticipantResolverContext into every call it makes to ParticipantResolverAdapter.

Extends EventROContext.

EventROContext

Provides read/write access to the managed objects in Identity Manager events. This interface is part of the core API functional model.

Extends IMContext.

The above components are in the package com.netegrity.imapi.

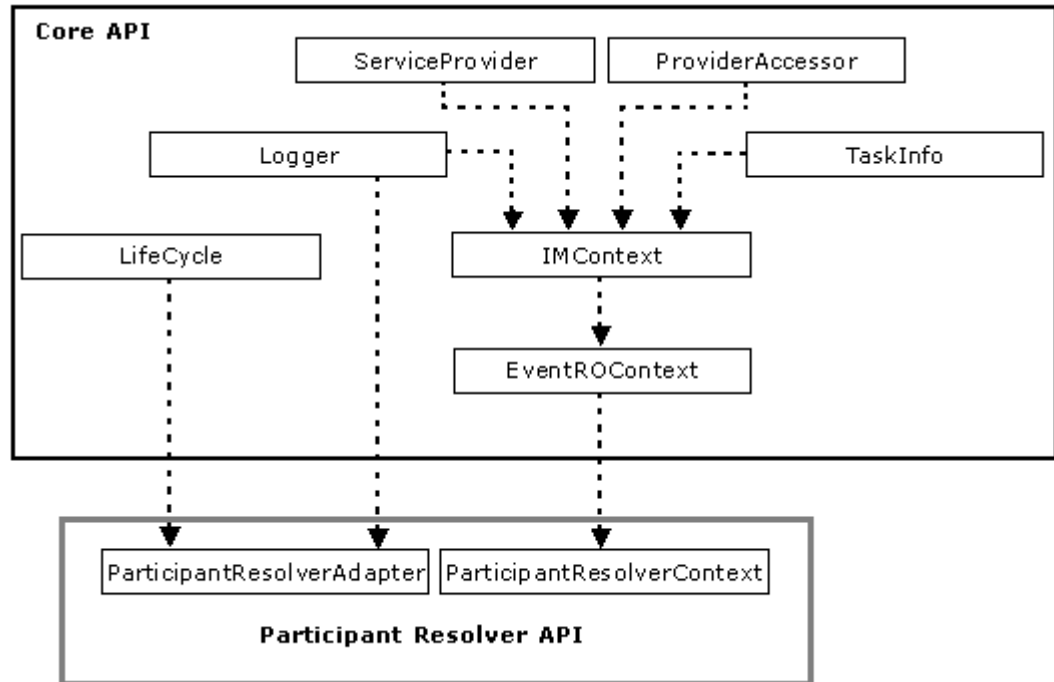
Note: In IdentityMinder v6.0, the objects ParticipantResolverAdapter and ParticipantResolverContext replaced the obsolete objects OrgResolverAdapter and OrgResolverContext. Custom resolvers that use these obsolete objects are no longer supported.

More Information:

[Access to Event Objects](#) (see page 124)

Participant Resolver API Model

The components of the Participant Resolver API are part of the Identity Manager API functional model:



Sample Participant Resolver

A sample implementation of a participant resolver is installed in the samples\ParticipantResolver subdirectory of your Identity Manager installation—for example:

C:\Program Files\CA\Identity Manager\samples\ParticipantResolver

Note: For information about using the sample, see the readme.txt file that accompanies the sample.

Calling Sequence

When the workflow engine needs to find an activity's participants, it passes the property information defined within the workflow engine's user interface to Identity Manager. This information includes the type of participant resolver to use (as listed below, by order of precedence), the name of the participant resolver (with custom types), and optional user-defined data. Identity Manager then does the following:

- Calls the appropriate participant resolver, based on the specified participant type.
- Retrieves the participant list and passes it back to the workflow engine.

If multiple participant resolver types have been specified, Identity Manager selects one based on the following order of precedence:

- Custom type
- Role type
- Filter type
- Group type

Identity Manager uses the first participant resolver it finds and ignores the rest.

Note: The workflow engine interacts with the Participant Resolver API indirectly, through an Identity Manager-supplied script called IM Approvers. This script passes Identity Manager all the relevant property information defined within the workflow engine's user interface, and Identity Manager passes the information to the Participant Resolver API. This script name must be included in the Resources properties of any workflow activity that requires an approval of an Identity Manager task.

Note: For more information, see the *CA Identity Manager Administration Guide*.

Participant Resolver Configuration

You define a participant resolver's fully qualified class name and other properties in the Identity Manager Management Console.

To configuring a participant resolver:

1. In the Management Console, click Identity Manager Environments.
2. Click the name of the environment where the participant resolver will be used.
3. Click Advanced Settings.
4. Click Workflow Participant Resolvers.

From the Workflow Participant Resolvers screen, you can do the following:

- Register a new participant resolver with Identity Manager. To begin this operation, click New.
- Modify an existing participant resolver. To begin this operation, click the participant resolver name.
- Delete a participant resolver.

For detailed instructions about performing these operations, click the Help link on the Workflow Participant Resolvers screen.

More Information:

[Compiling and Deploying](#) (see page 145)

Chapter 6: Notification Rule API

This section contains the following topics:

[API Overview](#) (see page 103)

[Email Templates](#) (see page 104)

[API Components](#) (see page 105)

[Notification Rule Configuration](#) (see page 107)

API Overview

The Notification Rule API lets you determine the users who should receive an email notification regarding an Identity Manager event or task. An email notification's text and recipient list are generated from an email template.

Identity Manager is shipped with three predefined notification rules:

ADMIN

Sends the email to the administrator who initiated the task.

USER

Sends the email to the user in the current event context.

USER_MANAGER

Sends the email to the manager of the user in the current context.

Notification Rule API Summary

Operation

Identity Manager Generates Email Notifications.

Called by

The `_util.getNotifiers()` method in any email template.

When called

An email notification is being generated from a template.

Operates on

Identity Manager objects.

Object update

Not relevant.

Can validate?

Not relevant.

Use Case Examples

Examples of how the Notification Rule API can be used:

Notify a single user

When user data is being modified, retrieve the user's name and cc the user with the status of the request.

Notify a group

When a user is being created and added to a group, retrieve the group name (from the context information passed into the notification rule) and cc the group about the new group member.

Email Templates

A notification rule determines the recipients of an email that is generated for an Identity Manager operation.

Identity Manager generates email automatically, based upon an email template. You use the Email Template API to customize an email notification and provide it with task-specific information.

The Email Template API contains the following objects in the package `com.netegrity.imapi`:

- `ExposedEventContextInformation`
- `ExposedTaskContextInformation`

These objects can be accessed directly from an email template through JavaScript. Custom Java objects do not have to be created and compiled.

Note: For more information about using these objects in an email template, see the *CA Identity Manager Administration Guide*.

API Components

The Notification Rule API contains the following components.

The list includes the core object EventROContext and a description of how it is used with this API:

NotificationRuleAdapter

The base class that all notification rules implement. Contains base implementation methods for startup and shutdown operations. Also returns the list of email addresses to receive the notification.

Implements LifeCycle and Logger.

NotificationRuleContext

Contains user-defined data passed from the email template.

This interface also provides access to IMContext information about the current task, and to logging methods and other Identity Manager services, including managed object retrieval from the data store.

A NotificationRuleContext object is passed to the methods of the NotificationRuleAdapter object.

Extends EventROContext.

EventROContext

Provides read/write access to the managed objects in Identity Manager events. This interface is part of the core API functional model.

Extends IMContext.

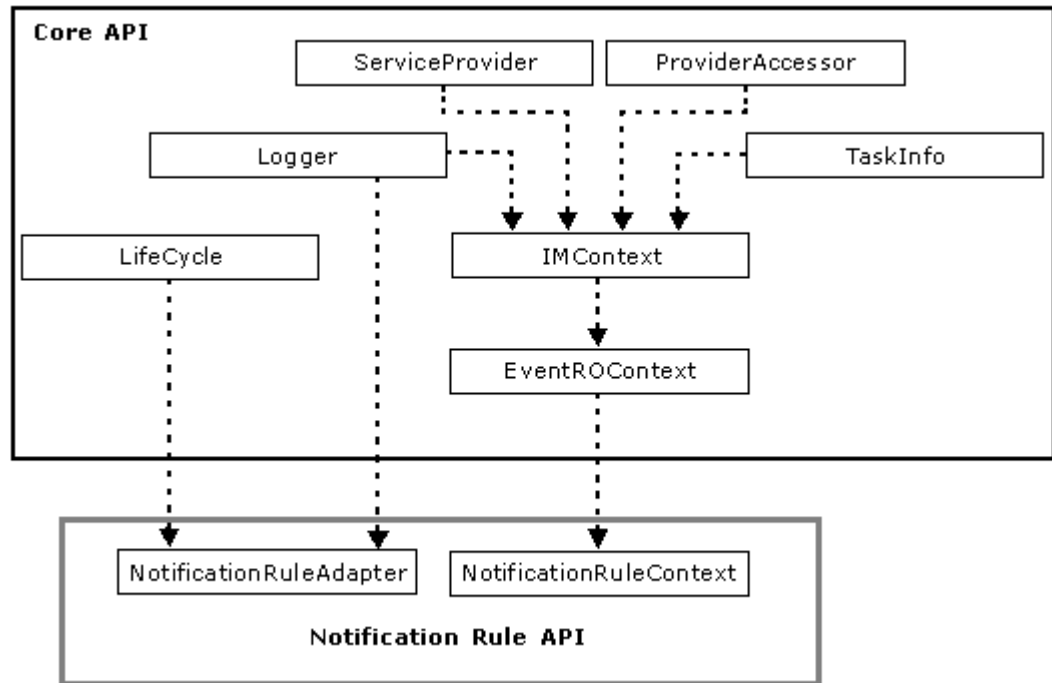
The above components are in the package com.netegrity.imapi.

More Information:

[Access to Event Objects](#) (see page 124)

Notification Rule API Model

The components of the Notification Rule API are part of the Identity Manager API functional model:



Sample Notification Rule

A sample implementation of a notification rule is installed in the `samples\NotificationRuleAdapter` subdirectory of your Identity Manager installation—for example:

`C:\Program Files\CA\Identity Manager\samples\NotificationRuleAdapter`

Note: For information about using the sample, see the `readme.txt` file that accompanies the sample.

Notification Rule Configuration

You define a notification rule's fully qualified class name and other properties in the Identity Manager Management Console. To do so, navigate to the Notification Rules screen, as follows:

1. In the Management Console, click Identity Manager Environments.
2. Click the name of the environment where the notification rule will be used.
3. Click Advanced Settings.
4. Click Notification Rules.

From the Notification Rules screen, you can:

- Register a new notification rule with Identity Manager. To begin this operation, click New.
- Modify an existing notification rule. To begin this operation, click the notification rule name.
- Delete a notification rule.

For detailed instructions about performing these operations, click the Help link on the Notification Rules screen.

Chapter 7: Managed Objects

This section contains the following topics:

[Managed Object Overview](#) (see page 109)

[Managed Object Interfaces](#) (see page 110)

[Access to Managed Object Data](#) (see page 114)

[Access to Objects in the Data Store](#) (see page 117)

[Access to Objects in a Task](#) (see page 121)

[Access to Event Objects](#) (see page 124)

Managed Object Overview

Identity Manager administrators perform admin tasks against managed objects. The Identity Manager APIs let custom objects manage or modify managed objects to satisfy the particular business requirements of the site.

Managed objects can be any of the following kinds of objects:

- Objects defined in the Identity Manager Directory for a particular Identity Manager environment (User, Group, Organization).

An Identity Manager Directory defines a logical view of users, groups, and organizations whose physical definitions are stored in a user directory. For each of the logical objects, a single Identity Manager object-class is defined.

The Identity Manager Directory is an extension of a SiteMinder Directory.

- Proprietary objects defined within an Identity Manager environment (Admin Role, Admin Task, Access Role, Access Task, Password Policy).
- Objects defined in the eTrust Admin Directory associated with the Identity Manager environment (ProvisioningRole).

A managed object contains a set of attributes that define the object's *profile*. For example, a User managed object would include the user's name, address, title, password settings, enabled state, and other information about the user.

Managed Object Interfaces

The following interfaces for Identity Manager managed objects are accessible through Identity Manager APIs.

All interfaces are in this package:

```
com.netegrity.llsdk6.imsapi.managedobject
```

AccessRole

Represents an Identity Manager access role. An *access role* lets role members perform specified tasks in applications other than Identity Manager. Access roles are available for SiteMinder Role-Based Access Control (RBAC) policies.

AccessTask

Represents an access task. An *access task* is a business function performed in an application other than Identity Manager. Access tasks are included in access roles.

Note: Identity Manager provides administrative control of access tasks, but it does not execute these tasks.

AdminRole

Represents an admin role. An *admin role* allows role members to perform specified administrative tasks in Identity Manager—for example, management of users, groups, organizations, roles, and tasks.

AdminTask

Represents an admin task. An *admin task* is an Identity Manager administrative task such as creating a user or a group. Admin tasks are included in admin roles.

Group

Represents a logical association of users in a user directory. Group members can be from multiple organizations.

Organization

Represents a container object in a user directory. It can contain other managed objects, including other organizations. Organizations typically correspond to the business units in an enterprise.

PasswordPolicy

Represents a set of restrictions and controls around passwords.

ProvisioningRole

Represents the definition of an eTrust Admin provisioning role to Identity Manager.

This object requires an instance of an eTrust Admin directory.

Role

Represents a logical grouping of administrative or business tasks. The managed object interfaces AdminRole, AccessRole, and ProvisioningRole implement the Role interface.

This interface includes the method modifyObject(Task[]). This method lets you commit changes to a managed object's attributes, just as with method modifyObject(). In addition, modifyObject(Task[]) lets you specify the complete set of tasks to associate with the role. You can specify the role's task set whether or not there are any attribute changes to commit.

Task

Represents an administrative or business function. The managed object interfaces AdminTask and AccessTask implement the Task interface.

User

Represents an individual within an organization in a user directory.

Note: For detailed information, see the Javadoc reference.

More Information:

[Super Interfaces](#) (see page 111)

Super Interfaces

All Identity Manager managed objects have the following super interfaces:

ManagedObject

Lets you access a managed object that extends this interface.

Package: com.netegrity.llsdk6.imsapi.managedobject

ModifiableObject

Lets you commit to the data store any changes made to the local (in-memory) attribute set of the current managed object. Changes are committed through the method modifyObject().

Package: com.netegrity.llsdk6.imsapi.abstractinterface

AttributeCollection

Provides access to managed object attributes as name/value pairs. The attributes you can access correspond to the fields and values on an Identity Manager task screen for a given managed object.

This interface includes methods that restrict access to attributes according to the permissions, if any, in the managed object.

Package: com.netegrity.llsdk6.imsapi.abstractinterface

NamedObject

Retrieves the unique and human-readable names of a managed object.

Package: com.netegrity.llsdk6.imsapi.abstractinterface

Extended Interfaces

All managed objects extend ManagedObject, either directly or through Role and Task.

Group and User objects extend the following additional interfaces:

Group objects extend the following interface:

- **GroupableObject**—defines an object that can become a member of a group.

User objects extend the following interfaces:

- **AssignableObject**—defines an object that can become a member of a role.
- **Grantor**—defines an object that can assign other objects to a role.
- **GroupableObject**—defines an object that can become a member of a group.

Managed Object Attributes

Identity Manager managed objects extend `ManagedObject`, and `ManagedObject` extends `AttributeCollection`.

The methods in `AttributeCollection` allow a custom object to retrieve and set managed object attributes.

You access a managed object's attributes using any of the following identifiers:

- Attribute constants defined in each managed object.
- Physical or well-known attribute names. The well-known attribute names are defined as the `PROPERTY_` constants in the `User`, `Group`, and `Organization` interfaces—for example:
`User.PROPERTY_EMAIL`
- Logical attribute names. When referencing a logical attribute, enclose the name in vertical bars (`|logicalAttributeName|`).

More Information:

[Retrieving Event Objects](#) (see page 125)

Retrieval Example

In the following example, a managed object is retrieved from an event. If the object's PROPERTY_EMAIL attribute does not already contain a user's email address, a default email address is created:

```
public int before(EventContext evtCtx) throws Exception {
    IMEvent evt = evtCtx.getEvent();
    if (evt instanceof UserEvent) {
        User user = ((UserEvent) evt).getUser();
        try {
            String userMail =user.getAttribute(User.PROPERTY_EMAIL);
            logDebugMessage("User Specified EMAIL "+userMail,true);
            logDebugMessage("Default EMAIL "+user.getFriendlyName()
                + email, true);
            if(userMail == null || userMail.length() == 0 ) {
                user.setAttribute(User.PROPERTY_EMAIL,
                    user.getFriendlyName() + email);
            }
        } catch (Exception ex) {
            logDebugMessage("Set EMAIL exception: "+ex.getMessage()
                + " in event " + evt.getEventName(), true);
        }
    }
    return CONTINUE;
}
```

Access to Managed Object Data

Managed object data is available to all Identity Manager APIs—for example:

- To the Business Logic Task Handler API, when validating user-supplied data on a task screen against attributes of Identity Manager objects
- To the Workflow API, when using attributes of Identity Manager objects to determine the next activity to perform in a workflow process
- To the Event Listener API, when performing these actions:
 - Setting or retrieving attributes of Identity Manager objects during the following event states: Pre-approval, Approved, and Rejected
 - Retrieving attributes of Identity Manager objects during the Post-execution event state

You can access managed objects in these ways:

ProviderAccessor

Interface

ProviderAccessor

API

All

Comments

Read/write access.

Provides direct access to the information in the data store.

All APIs can read attribute data in objects retrieved through ProviderAccessor, but typically only business logic task handlers and event listeners modify these objects.

Modifications are committed to the data store after the custom object calls modifyObject(). The modifications are committed immediately. No events are generated, and no workflow approvals, auditing, or security checks are performed.

BLTHContext

Interface

BLTHContext

API

Business Logic Task Handler

Comments

Read/write access.

Provides access to the run time instance of objects in the current task. Objects are accessed through BLTHContext get... methods.

Identity Manager commits changes to the data store after any workflow approvals have been provided and the execution of events associated with the task is complete.

EventContext

Interface

EventContext

API

Event Listener

Comments

Read/write access. Can also generate secondary events.

Provides access to the run time instance of an object in a particular event.

Identity Manager commits changes to the data store after any workflow approvals have been provided and the execution of the event is complete.

EventROContext

Interface

EventROContext

API

Notification Rule Participant Resolver

Comments

Read/write access.

Provides access to the run time instance of an object in a particular event.

Identity Manager commits changes to the data store after any workflow approvals have been provided and the execution of the event is complete.

More Information:

[Access to Objects in the Data Store](#) (see page 117)

[Access to Objects in a Task](#) (see page 121)

[Access to Event Objects](#) (see page 124)

Access to Objects in the Data Store

All APIs can access managed object data in the data store. You access managed objects in the data store through the `ProviderAccessor` interface.

Typically, managed objects retrieved through `ProviderAccessor` are accessed for reading purposes only. However, on occasion, business logic task handlers and event listeners may need to modify objects retrieved through `ProviderAccessor`.

Note: If you modify the attributes of a managed object retrieved through `ProviderAccessor`, you are modifying the local (in-memory) representation of the object. To persist the changes to the data store, call `modifyObject()`, which is inherited from `ModifiableObject`. The changes are then made directly to the data store. No Identity Manager events are generated, and no workflow approvals, auditing, or security checks are performed.

Providers

A *provider* is an object that provides access to managed objects and other Identity Manager objects in the data store.

Provider objects include methods for performing data store operations such as:

- Creating an object
- Retrieving a particular object by name
- Retrieving one or more objects according to some search criteria

Provider objects are located in the following package:

`com.netegrity.llsdk6.imsapi.provider`

Identity Manager includes the following providers:

- AccessControlProvider
- AccessRoleProvider
- AccessTaskProvider
- AdminRoleProvider
- AdminTaskProvider
- GroupProvider
- OrganizationProvider
- ProvisioningPolicyProvider
- ProvisioningRoleProvider
- SecurityProvider
- SynchronizationProvider
- UserProvider

Retrieving Providers

ProviderAccessor is one of the core objects in the Identity Manager API functional model. In each API, the ...Context object (such as BLTHContext, LogicalAttributeContext) inherits ProviderAccessor methods. In addition, ProviderAccessor can be made available to custom objects when the Identity Manager environment is initialized.

You retrieve provider objects through ProviderAccessor get... methods. For example, to access a UserProvider object, call getUserProvider(), and to access an AdminTaskProvider object, call getAdminTaskProvider().

When you update an object retrieved through ProviderAccessor, no Identity Manager events are generated because the managed object is not part of a task session. Managed object updates are reflected in Identity Manager events only if the updates are made to a managed object in a task session.

More Information:

[Access to Objects in a Task](#) (see page 121)

Attribute and Permission Requests

When you retrieve a managed object through a provider, you can specify the following:

- The attributes to include in the retrieved object
- A permission request for each attribute

You specify attribute and permission requests in `AttributeRightsCollection`.

`AttributeRightsCollection` is a collection of `AttributeRight` objects.

Permission Request Rules

Identity Manager calculates the access permission for a particular attribute based upon information in the current task context and information stored on the server—namely, upon the attribute's permissions for all roles that contain the current task and that involve the current administrator and operation.

If a permission request specified in `AttributeRightsCollection` differs from the calculated permission for a given attribute, the following rules apply:

- If a permission request is more restrictive than the calculated permission, the requested permission will apply to the attribute. For example, if you pass in a `READONLY` request for a `READWRITE` attribute, the `READONLY` permission is used.

The requested permission applies over the calculated permission only in the current instance of the managed object.

- If a permission request is less restrictive than the calculated permission, the calculated permission is used.

Access to Attribute Data

`AttributeCollection` allows access to the data in managed object attributes.

`AttributeCollection` contains methods that restrict access to attributes according to the access permissions associated with the attributes. For example, if you call `setAttribute()` for an attribute whose associated permission does not allow write access, an `AttributePermissionException` is thrown.

Calling Sequence

To access a managed object through ProviderAccessor:

1. Call the ProviderAccessor get... method that retrieves the provider for the type of managed object you want to access. For example, to access a User managed object, call getUserProvider() to retrieve a UserProvider object.
2. Use the provider to create a managed object or to retrieve one or more managed objects. For example, if you want to work with a User managed object, you could make a call such as:
 - Calling createUser() to create a User object in a particular organization and with a particular set of attributes.
 - Calling findUser() to retrieve a specified User object.
 - Calling getUsersInOrg() to retrieve all the User objects in a specified organization.
3. Once you have a managed object, access the object's attributes through AttributeCollection.
4. If you create a managed object or update a managed object's attributes, you are acting on the local copy of the object. To persist the newly assigned or modified attribute values to the data store, call modifyObject(). Managed objects inherit this method from ModifiableObject.

Note: If you are assigning attribute values for a role object, you can define the complete set of tasks to associate with the role and commit the attribute changes to the data store in the same call. You do so with the method modifyObject(Task[]) in the Role interface.

More Information:

[Managed Object Attributes](#) (see page 113)

Access to Objects in a Task

Business logic task handlers can access managed objects and related information in the current task session.

A task session includes the following kinds of information:

- The managed object that is the subject of the task. For example, the subject of a Modify User task is a User managed object.

The subject can be a Vector of objects in the following circumstances:

- If the task involves deleting User, Group, or Organization objects
- If the task involves enabling or disabling User objects
- The subject's relationship objects. A *relationship* indicates a resource to which the subject is assigned.

For example, with a Modify User task, the User object's relationships can include any of the user's group and role assignments.

Parallels with Task Screen Operations

A task's subject and its relationships are represented by tabs on a task screen. For example, a Create User task might have the following tabs:

- User Profile tab (subject)
- User Access Roles tab (relationship)
- User Admin Roles tab (relationship)
- User Groups tab (relationship)

A business object task handler's access to managed objects in a task is similar to a user's access to managed objects in a task screen tab. In both cases, access is to a run time instance of the managed object data in the task session. There is no direct access to the data in the data store, as is the case when accessing managed objects through the providers.

By retrieving the subject of the task and the subject's relationships, a business logic task handler can perform operations similar to a user working in a task screen—for example:

- Modifying the profile information for the subject of the task (such as modifying the phone number, job title, and email address in a user profile).
- Adding or removing the subject's relationships (such as adding the user to a group or removing the user from a role).

Retrieval of Managed Objects

To retrieve a managed object in a task session, call one of the get... methods in `BLTHContext`, for example:

- Call `getUser()` to retrieve the user object that is the subject of the current task.
- Call `getAdminTask()` to retrieve the admin task being executed in the current task.
- Call `getGroupMembers()` to retrieve the group members who are being added to or removed from the group in the current task.
- Call `getEnabledUsers()` or `getDisabledUsers()` to retrieve the user objects enabled or disabled in the current task.

Operations on Managed Objects

When managed objects are modified in the task session, either through a business logic task handler or a task screen, Identity Manager performs the following operations when the task enters the asynchronous phase of task processing:

- Generates events for the modifications to the subject object and its relationships. The events can be subject to workflow approval, auditing, and security checks.

For example, if a business logic task handler adds a user to a group, Identity Manager generates `AddToGroupEvent`, and the event can be subject to workflow approval.

Note: When you update a managed object in a task session, the changes are reflected in the events that Identity Manager generates for the task. However, if you update a managed object retrieved through `ProviderAccessor` (rather than a managed object in the task session), the changes are not reflected in Identity Manager events.

- Commits the modifications to the data store after events associated with the task have been executed. You do not call `modifyObject()` to persist the modifications.

Event Generation

When you retrieve a managed object through a `BLTHContext` `get...` method and then update the object, the update is represented by one or more events that Identity Manager generates. For example:

- If you change the email address and telephone number in an existing user's profile (the subject of the task), the changes are reflected in the event `ModifyUserEvent`.
- If you add the subject of the task to five groups, five `AddToGroupEvent` events are generated. This allows a separate workflow process and approval to be associated with each group assignment.

More Information:

[Access to Event Objects](#) (see page 124)

Calling Sequence

To access and update a managed object in a task session:

1. Call one of the `get...` methods in `BLTHContext` to retrieve the managed object to be updated.
2. Manage the task's subject and its relationships through `AttributeCollection`.

More Information:

[Managed Object Attributes](#) (see page 113)

Attribute Validation Type

There are two types of validation rules: task-level and directory-level.

If a managed object attribute is associated with a validation rule, Identity Manager applies the rule when the attribute value is set.

Identity Manager enforces validation rules when a business logic task handler calls a `setAttribute...` method in `AttributeCollection` to set an attribute. However, Identity Manager will only enforce directory-level validation for attributes set in this way. That is because, when `setAttribute...` is called, the attribute value has already been set in the task screen field, and is currently being set in a managed object in the task session.

More Information:

[Validation Objects](#) (see page 142)

Access to Event Objects

When an Identity Manager task reaches the asynchronous phase (that is, after the user clicks Submit on the task screen and the user-supplied data has been validated), Identity Manager breaks down the task into its component events.

These events can be accessed by custom objects that Identity Manager calls during the asynchronous phase, and by workflow scripts that call the methods in the Workflow API.

Event objects are accessed through one of the following interfaces. Each interface allows read/write access to a managed object in an Identity Manager event. EventContext also allows the generation of secondary events:

EventContext

- Extends IMPersistentContext, which extends IMContext.
- EventContext allows information to be persisted throughout the life of an event.
- Extended by WorkflowContext.

EventContext is also the ...Context object for the Event Listener API.

EventROContext

- Extends IMContext.
- Extended by ExposedEventContextInformation, NotificationRuleContext, ParticipantResolverContext.

If you modify a managed object in an event, Identity Manager commits the modifications to the data store after any approvals required for the execution of the event have been provided and the execution of the event is complete. You do not call `modifyObject()` to persist the modifications.

Events and Event Objects

Every Identity Manager event is represented by an event object. An event object has the same name as the corresponding event.

Identity Manager generates an object for each event and makes the objects available for retrieval through EventContext and EventROContext. Once you retrieve an event object, you can retrieve the managed object associated with the event object.

Retrieving Event Objects

EventContext and EventROContext let you access the current event object. You can do so in either of the following ways, as illustrated in the following code for an event listener:

- Call `getEventName()` to return the name of the current event. You can test for the events that the event listener is mapped to, and that you expect to apply to this listener, and then retrieve the event object accordingly:

```
int before(EventContext eCtx) {
    if(eCtx.getEventName().equals(ImEventName.CREATEUSEREVENT))
        CreateUserEvent evt = (CreateUserEvent);
    . . .        // Event listener processing continues.
}
```

Note: For a list of event name constants, see `ImEventName` in the Javadoc reference.

- Call `getEvent()` to return an `IMEvent` base interface for the event. For example, if the event listener is mapped to a specific event, such as `CreateUserEvent`, you can cast the `IMEvent` object as a `CreateUserEvent` object, as follows:

```
int before(EventContext eCtx) {
    CreateUserEvent evt = (CreateUserEvent)eCtx.getEvent();
    User user = evt.getUser();
    user.setAttribute("CreateTime",new Date().toString());
}
```

More Information:

[Events and Event Objects](#) (see page 124)

Base Event Interfaces

Each Identity Manager event object implements one or at most two of the base event interfaces listed in the following table.

Most base event interfaces contain a single `get...` method, which retrieves the managed object contained in the event. For example, `AccessRoleEvent` contains a `getAccessRole()` method that returns an `AccessRole` managed object. The exception is `ExternalTaskEvent`. Because external tasks are executed by third-party applications, `ExternalTaskEvent` contains no `get...` method for retrieving a managed object.

Once a custom object or a workflow script retrieves the current event object, the `get...` methods that the event object inherits from the base interfaces can retrieve the managed objects contained in the event.

Base Interface	Method	Managed Object Retrieved
AccessRoleEvent	getAccessRole()	AccessRole
AccessTaskEvent	getAccessTask()	AccessTask
AdminRoleEvent	getAdminRole()	AdminRole
AdminTaskEvent	getAdminTask()	AdminTask
GroupEvent	getGroup()	Group
OrganizationEvent	getOrganization()	Organization
ParentOrganizationEvent	getParentOrganization()	Organization
ProvisioningRoleEvent	getProvisioningRole()	ProvisioningRole
UserEvent	getUser()	User

Note: Base event interfaces are not Identity Manager events. Identity Manager events implement the base event interfaces.

The get... methods that a particular event object inherits are determined by the base interfaces that the event object implements. For example:

- A CreateAdminRoleEvent object implements a single base interface—AdminRoleEvent. The getAdminRole() method that CreateAdminRoleEvent inherits allows a custom object or script to retrieve an AdminRole managed object.
- A CreateUserEvent object implements the base interfaces UserEvent (for the user being created) and ParentOrganizationEvent (for the organization where the user is being created). Therefore, the CreateUserEvent object inherits the following methods:
 - getUser(), for retrieving a User managed object
 - getParentOrganization(), for retrieving the Organization managed object

Custom Audit Event Implementation

Audit data provides a historical record of operations that occur in an Identity Manager environment. You can enable auditing for some or all of the Identity Manager events generated by admin tasks.

Note: For information about configuring auditing in your environment, see the *CA Identity Manager Configuration Guide*.

You can implement application-specific auditing in cases where you might have custom code that performs a non-standard operation—for example, an object that updates a user directory directly, rather than through Identity Manager. You could create a custom audit event to report unexpected results back to Identity Manager.

To create an audit event to support your custom application, write a class that implements `AuditEvent` and that extends `GenericAuditEvent`. Typically, an extension of this class will update the description associated with the custom event, which is passed to Identity Manager when the event occurs. You can override that class's `extractAuditData()` method to process the audit data passed in to it (an `EnhancedAuditData` object) as appropriate for the situation.

The auditable attributes of an event are contained in `EventProfile`. This class also lets you set and retrieve the stage of an event cycle in which event attributes are recorded for auditing.

You can find a sample BLTH application with an associated audit event in the following directory:

`C:\Program Files\CA\Identity Manager\samples\BLTH\Group`

Event Description Localization

A description of an event is presented to users at various times—for example:

- When users view the tasks they submitted for processing, by executing View My Submitted Tasks. This task lists the events that are generated by each submitted task, along with the descriptions of those events.
- When a workflow work item is presented to a user. The work item information includes the event and event description.

Default event descriptions are defined in the base resource bundle `IMSResources.properties`.

Note: For information about using this file for localization, see the *CA Identity Manager Configuration Guide*.

Event Object Tables

The following tables show the base interfaces that each event object implements. The tables help you determine the get... methods and consequently, the managed objects contained in the current event object.

See the referenced section for a listing of the get... method and managed object associated with each base interface.

More Information:

[Base Event Interfaces](#) (see page 125)

User Events Table

Identity Manager Event	Implemented Base Interface
CreateUserEvent	UserEvent ParentOrganizationEvent
DeleteUserEvent	UserEvent ParentOrganizationEvent
ModifyUserEvent	UserEvent ParentOrganizationEvent
ViewUserEvent	UserEvent ParentOrganizationEvent
ForgottenPasswordEvent	UserEvent ParentOrganizationEvent
ForgottenUserIDEvent	UserEvent ParentOrganizationEvent
ResetPasswordEvent	UserEvent ParentOrganizationEvent
DisableUserEvent	UserEvent ParentOrganizationEvent
EnableUserEvent	UserEvent ParentOrganizationEvent

Organization Events Table

Identity Manager Event	Implemented Base Interface
CreateOrganizationEvent	OrganizationEvent ParentOrganizationEvent
DeleteOrganizationEvent	OrganizationEvent ParentOrganizationEvent
ModifyOrganizationEvent	OrganizationEvent ParentOrganizationEvent
ViewOrganizationEvent	OrganizationEvent ParentOrganizationEvent

Group Events Table

Identity Manager Event	Implemented Base Interface
CreateGroupEvent	GroupEvent ParentOrganizationEvent
DeleteGroupEvent	GroupEvent ParentOrganizationEvent
ModifyGroupEvent	GroupEvent ParentOrganizationEvent
ViewGroupEvent	GroupEvent ParentOrganizationEvent
AddToGroupEvent	UserEvent GroupEvent
RemoveFromGroupEvent	UserEvent GroupEvent
AddGroupAdminEvent	UserEvent GroupEvent
RemoveGroupAdminEvent	UserEvent GroupEvent

Task Events Table

Identity Manager Event	Implemented Base Interface
CreateAdminTaskEvent	AdminTaskEvent
DeleteAdminTaskEvent	AdminTaskEvent
ModifyAdminTaskEvent	AdminTaskEvent
ViewAdminTaskEvent	AdminTaskEvent
CreateAccessTaskEvent	AccessTaskEvent
DeleteAccessTaskEvent	AccessTaskEvent
ModifyAccessTaskEvent	AccessTaskEvent
ViewAccessTaskEvent	AccessTaskEvent

Role Events Table

Identity Manager Event	Implemented Base Interface
CertifyRoleEvent	Role (Admin, Access, Provisioning)
CreateAdminRoleEvent	AdminRoleEvent
DeleteAdminRoleEvent	AdminRoleEvent
ModifyAdminRoleEvent	AdminRoleEvent
ViewAdminRoleEvent	AdminRoleEvent
CreateAccessRoleEvent	AccessRoleEvent
DeleteAccessRoleEvent	AccessRoleEvent
ModifyAccessRoleEvent	AccessRoleEvent
ViewAccessRoleEvent	AccessRoleEvent
CreateProvisioningRoleEvent	ProvisioningRoleEvent
DeleteProvisioningRoleEvent	ProvisioningRoleEvent
ModifyProvisioningRoleEvent	ProvisioningRoleEvent
ViewProvisioningRoleEvent	ProvisioningRoleEvent

Role Assignment Events Table

Identity Manager Event	Implemented Base Interface
AssignAccessRoleEvent	UserEvent
RevokeAccessRoleEvent	UserEvent
AssignAdminRoleEvent	UserEvent
RevokeAdminRoleEvent	UserEvent
AssignProvisioningRoleEvent	UserEvent
RevokeProvisioningRoleEvent	UserEvent

Role Grant Events Table

Identity Manager Event	Implemented Base Interface
AddGrantorOnAdminRoleEvent	UserEvent
RemoveGrantorOnAdminRoleEvent	UserEvent
AddGrantorOnAccessRoleEvent	UserEvent
RemoveGrantorOnAccessRoleEvent	UserEvent
AddGrantorOnProvisioningRoleEvent	UserEvent
RemoveGrantorOnProvisioningRoleEvent	UserEvent

Self Registration Event Table

Identity Manager Event	Implemented Base Interface
SelfRegisterUserEvent	UserEvent ParentOrganizationEvent

Synchronization Events Table

Identity Manager Event	Implemented Base Interface
SynchronizeUserAccountEvent	UserEvent ParentOrganizationEvent
SynchronizeUserEvent	UserEvent ParentOrganizationEvent
SynchronizeUserProvisioningRoleAddAccountsEvent	UserEvent
SynchronizeUserProvisioningRoleAddAccountsEvent	UserEvent

User Certification Events Table

Identity Manager Event	Implemented Base Interface
CertificationNonCertifiedActionCompletedNotificationEvent	UserEvent
CertificationNonCertifiedActionPendingNotificationEvent	UserEvent
CertificationRequiredFinalReminderNotificationEvent	UserEvent
CertificationRequiredNotificationEvent	UserEvent
CertificationRequiredReminderNotificationEvent	UserEvent
CertificationStatusCertifiedEvent	UserEvent
CertificationStatusInCertificationEvent	UserEvent
CertificationStatusNotCertifiedEvent	UserEvent
CertificationStatusRequiredCertificationEvent	UserEvent

Chapter 8: Support Objects

This section contains the following topics:

[Identity Policies](#) (see page 133)
[IMEvent](#) (see page 134)
[IMEventName](#) (see page 134)
[IMSEException](#) (see page 134)
[Localizer Class](#) (see page 136)
[PasswordCondition](#) (see page 137)
[Permissions Objects](#) (see page 137)
[Policies and Policy Conditions](#) (see page 138)
[PropertyDict](#) (see page 138)
[Provisioning Roles](#) (see page 139)
[ResultsContainer](#) (see page 139)
[RoleDisplayData](#) (see page 140)
[Search Objects](#) (see page 140)
[Task.FieldId](#) (see page 141)
[TSContext](#) (see page 141)
[Type Objects](#) (see page 141)
[Validation Objects](#) (see page 142)

Identity Policies

Identity policies define rule-based provisioning behaviors. These policies map a member rule to a set of rules for allocation and deallocation. For example, when a user is promoted and takes on the title Vice President, Identity Manager could automatically add this person to the group Country Club Member, and also automatically assign the Role Salary Reviewer.

In the context of identity policies, synchronization refers to the process of evaluating what policies a user matches, comparing that list to the stored set of policies that have already been allocated, and determining which policies must be allocated or deallocated by comparing the two lists.

The following objects in part manage synchronizing users with identity policies:

- AutoSynchType
- IdentityPolicySetEvent
- SynchronizationProvider

Note: Some code element names may be based on the obsolete term "business policies" instead of "identity policies."

IMEvent

IMEvent is a base interface for an event. A custom object or workflow script can retrieve the IMEvent object for the current event, and then cast the object as the event.

Package

com.netegrity.imapi

More Information:

[Events and Event Objects](#) (see page 124)

IMEventName

The IMEventName interface contains constants for the names of Identity Manager events (typically, the constant name is the same as the corresponding event name, but in upper-case letters).

Package

com.netegrity.imapi

IMSEException

The IMSEException class contains one or more exception messages to be displayed to the user at run time. The messages in this object appear on a redisplayed task screen after the user submits the task screen and validation errors are found. This gives the user the opportunity to correct the errors and resubmit the task.

You populate this object with messages from files called *resource bundles*.

Package

com.netegrity.ims.exception

Localizing Exception Messages

The exception messages that you load into the IMSEException object are stored in resource bundles. Each resource bundle with the same base name contains the same messages, but in a different language.

Identity Manager includes a family of resource bundles containing Identity Manager system messages. These resource bundles have the base name IMSEExceptions.properties. By default, the system messages in the base resource bundle are in English. Optionally, you can translate the system messages in the base resource bundle into another language, but you should not modify the system messages in the base resource bundle for any other reason.

Message Format and Variables

Resource bundles are text files. The items in the resource bundles have the format *messageID=message*—for example:
5004=The SSN is a required field.

You can store custom exception messages in a custom resource bundle or in IMSEExceptions.properties. If you store custom messages in the IMSEExceptions.properties file, do so at the end of the file, and in the format indicated in the file. For example, you should prefix the message ID with custom-, as shown in the following line:
custom-5004=The SSN is a required field.

Using this format in the predefined IMSEExceptions.properties file helps to avoid migration problems when you update the Identity Manager software.

IMSEException messages support variables. You can insert specific values into the variables when you add the message to the IMSEException object.

Managing Custom Resource Bundles

You manage custom resource bundles as follows:

- Call getUserLocale() to determine the locale of the current user. This method is in the TaskInfo object that is passed to the business logical task handler context or logical attribute context.
- If you are using messages in a custom resource bundle, specify the base resource bundle name in the IMSEException constructor.
- To load a message into IMSEException, call addUserMessage() in the IMSEException object. The parameters include the locale and the unique code of the message to add.

The names of the resource bundles must follow a particular format:

- The base resource bundle name consists of the resource family name plus the extension `.properties`—for example:

`LocalizedMessages.properties`

- Other resource bundles consist of the base resource bundle name, an underscore, the two-character language identifier of the messages in the bundle, and the `.properties` extension. For example, a German resource bundle might be named:

`LocalizedMessages_de.properties`

Identity Manager supports the standard two-character ISO 639 language identifiers. You can view them in the following location:

<http://ftp.ics.uci.edu/pub/ietf/http/related/iso639.txt>

CA recommends that you store your custom resource bundles in the following location within the Identity Manager EAR directory on the application server:

`custom\resourceBundles`

If you choose not to store your custom resource bundles in `custom\resourceBundles`, you must store them in the custom directory or in any directory beneath it.

Note: For more information about localization, see the *CA Identity Manager Configuration Guide*.

Localizer Class

Localizer is a helper class for retrieving the localized version of a string for a given locale. For example, you can retrieve the localized version of a label on a task screen, a task name, or an exception message. The localized string is retrieved from the resource bundle associated with the Localizer object's locale.

Package

`com.netegrity.ims.util`

Note: For localization information, see the *CA Identity Manager Configuration Guide*.

More Information:

[IMSEException](#) (see page 134)

Use the Localizer Class

You can use the Localizer object from your custom code or from a JSP file for a new skin. You can localize the following JSP files for a new skin:

To localize a new skin

1. Add the localized strings to the IMSResources bundles provided with Identity Manager and to any custom bundles you might have. Assign each localized string to a unique property ID—for example:

```
skin.header.companyName=My Company
```

2. Use the Localizer class to extract a localized string. The string is retrieved from the resource bundle for the locale associated with the Localizer object. For example:

```
<%@ page import="com.netegrity.ims.util.*"%>
<% Localizer l10n = Localizer.getInstance();%>
<b><%=l10n.getLocalizedString("skin.header.companyName")%></b>
```

PasswordCondition

The PasswordCondition class encapsulates information about why a user password is invalid.

Package

```
com.netegrity.llsdk6.imsapi.utility
```

Permissions Objects

The permissions classes provide information about the access permission (for example, read-only or read/write) associated with a managed object attribute. The permissions classes are:

AttributeRight

The AttributeRight class contains the name of an attribute and its associated permission.

Package

```
com.netegrity.llsdk6.imsapi.metadata
```

AttributeRightCollection

The AttributeRightCollection class is a collection of AttributeRight objects. Use this object to specify the attributes (and their associated access permissions) to include in a managed object you are retrieving.

Package

com.netegrity.llsdk6.imsapi.collections

More information:

[Type Objects](#) (see page 141)

Policies and Policy Conditions

The following classes and interfaces let you manage admin policies, membership policies, and policy conditions:

- AdminPolicy
- MemberRule
- MembershipPolicy
- Policy
- ScopeRule
- TriggerRule

Note: Some code element names may be based on the obsolete term "trigger rules" instead of "policy conditions."

Packages

com.netegrity.llsdk6.imsapi.policy
com.netegrity.llsdk6.imsapi.policy.rule

PropertyDict

The PropertyDict interface contains a tab's configuration parameters as defined in the Identity Manager User Console.

Package

com.netegrity.llsdk6.imsapi.collections

Provisioning Roles

The following objects let you define eTrust Admin provisioning roles to Identity Manager, and to associate eTrust Admin provisioning roles with eTrust Admin provisioning policies from within Identity Manager:

- Account
- AccountHolder
- AccountReason
- ProvisioningPolicy
- ProvisioningPolicyProvider
- ProvisioningRole
- ProvisioningRoleProvider

Use of these objects requires an instance of an eTrust Admin directory.

Package

Various.

ResultsContainer

The ResultsContainer class provides information about the relationships of the subject of the current task.

For example, a task session for a Modify User task includes a User managed object for the administrator being modified. You can access information (such as name, title, email address, and so on.) about the administrator through the ProfileTabHandler. However, the task session might contain additional information about the administrator—for example, the administrator's group and role assignments made during the task session. This relationship information (the group and role objects) is available through ResultsContainer.

This class lets you view changes that occur to the objects contained within it during the execution of the task.

Package

com.netegrity.imapi

RoleDisplayData

The RoleDisplayData interface represents the information Identity Manager needs to display information about a role.

Package

`com.netegrity.llsdk6.imsapi.metadata`

Search Objects

Use the following classes to define and manage search operations in an Identity Manager environment.

- ApiResultSet
- RoleObjectQuery
- SearchCursor
- SearchExpression
- TaskObjectQuery

Package

`com.netegrity.llsdk6.imsapi.search`

Search Filters and Constraints

Use the following classes and interfaces to define search filters, membership constraints, and scope constraints:

- AttributeExpression
- GroupFilter
- MemberMatchConstraint
- OrgFilter
- OrgMembershipConstraint
- OrgMembershipIdentifier
- OrgScopeConstraint
- OrgScopeIdentifier
- RuleConstraint
- UserFilter

Package

`com.netegrity.llsdk6.imsapi.policy.rule.constraints`

Task.FieldId

The Task.FieldId class contains identifiers for the reserved fields on an access task configuration screen. The reserved fields let an administrator supply values that limit a user's access task scope.

Package

com.netegrity.llsdk6.imsapi.managedobject

TSContext

The TSContext interface contains information about the current task, such as the subject of the task, security information, and an instance of the task. TSContext is passed into methods of provider objects such as UserProvider.

Package

com.netegrity.llsdk6.imsapi.utility

Type Objects

The following type classes contain constants that define some aspect of a managed object or some operation involving a managed object:

AccountType

Contains constants for eTrust Admin Account objects.

ActionType

Contains constants that represent the type of operation being performed (for example, a modify, delete, or approval operation).

AttributeChangeType

Contains constants that represent the type of changes that should be audited.

AttributeExpressionType

Contains constants that indicate the type of attribute used in a search expression (for example, Admin attributes or primary object attributes).

ConjunctionType

Contains constants used in search expressions.

DisabledReasonType

Contains constants that indicate why a user account was disabled.

ObjectType

Contains constants that specify particular types of managed objects (for example, User, Org, Group objects).

OperatorType

Contains constants that specify the operators that can be used in search expressions.

PasswordMessageType

Contains constants that specify the reason a password is invalid.

PermissionType

Contains constants that represent the permissions assigned to directory attributes.

SearchDepthType

Contains constants that specify the scope of a search. For example, BASE indicates a search of the base organization only. N indicates a search of the base organization and all its children.

Package

com.netegrity.llsdk6.imsapi.type

Validation Objects

At run time, after a user enters a value in a task screen field, Identity Manager validates the value if the field is associated with one or more validation rules.

During task screen configuration, a task screen field can be associated with a validation rule in either of these ways:

- By directly associating a validation rule with a field on a task screen.

This type of attribute validation is called *task-level validation*. Identity Manager validates the field value against other attributes in the task—for example, validating an area code for a user's phone number against the user's city and state.

- By configuring a task screen field with a user directory attribute that is mapped to a validation rule set. The mapping between the attribute and the rules in the rule set is defined in `directory.xml`.

This type of attribute validation is called *directory-level validation*. Identity Manager validates the attribute value itself, rather than validating it in the context of other attributes in the task—for example, validating that a phone number matches the nnn-xxx-xxxx format used in the directory attribute.

Note: For more information, see the *CA Identity Manager Configuration Guide*.

Attribute Validation

The following objects are used during attribute validation:

AttributeValidator

The class for performing directory-level validation of managed object attributes. This abstract class is extended by directory-level validation rules that need to access user, group, and organization objects in the user directory.

Package

`com.netegrity.llsdk6.imsapi.metadata`

AttributeValidationException

The exception class for attribute validation errors.

Package

`com.netegrity.llsdk6.imsapi.exception`

DirectoryProvidersCollection

The interface for accessing the managed object providers during directory-level attribute validation. The managed object providers let you retrieve user, group, and organization objects from the user directory.

Package

`com.netegrity.llsdk6.imsapi.utility`

IAttributeValidator

The interface for performing directory-level validation of managed object attributes. This interface is implemented by directory-level validation rules. If you need to validate an attribute against information in the user directory, extend the AttributeValidator abstract class.

Package

com.netegrity.llsdk6.imsapi.metadata

IAttributeValidator.StringRef

A class that lets you return an updated attribute value or error message to Identity Manager during directory-level attribute validation.

Package

com.netegrity.llsdk6.imsapi.metadata

TaskValidator

Interface for performing task-level validation of managed object attributes. This interface is implemented by task-level validation rules.

Package

com.netegrity.imapi

TaskValidator.StringRef

A class that lets you return an updated attribute value or error message to Identity Manager during task-level attribute validation.

Package

com.netegrity.imapi

Chapter 9: Compiling and Deploying

This section contains the following topics:

[Compiling Source Files](#) (see page 145)

[Deploying Run Time Files](#) (see page 146)

Compiling Source Files

You can write Java objects that provide the following custom features:

- Logical attribute handlers
- Business logic task handlers
- Participant resolvers used in workflow operations
- Event listeners
- Email notification rules

Note: The sections on compiling do not apply to the Workflow API and to the Email Template API. You do not compile Java objects with these APIs. Instead, the methods in the Workflow API are called from workflow scripts, and the methods in implicit message objects are called from email templates.

When you write new .java files or modify existing ones, compile the .java files against the following .jar files:

ims.jar
imsapi6.jar

These files (along with Identity Manager API samples and other files) are installed when you select the Identity Manager Administrative Tools option during installation. The jar files are located in the Identity Manager lib directory. The default path is:

C:\Program Files\CA\Identity Manager\lib

You do not compile Java objects with the following APIs:

- Workflow API. These methods are called from workflow scripts
- Email Template API. These methods are called from email templates

Deploying Run Time Files

The following sections describe the locations where you should place custom run time files.

Note that the slash character is platform-specific. The windows version, which is the backslash (\), is used in the examples in the following sections. On UNIX platforms, use the forward slash (/).

Java Class Files

Identity Manager looks for compiled .class files in the following root location within your application server:

`IdentityMinder.ear\custom`

For example, if your custom object is `com.mycompany.myBLTH` and you are deploying on WebLogic, the following location is a valid location for `myBLTH.class`:

`C:\bea\user_projects\mydomain\applications\IdentityMinderr.ear\custom\com\mycompany\myBLTH.class`

JavaScript Files

If you implement a task-specific business logic task handler in JavaScript, and the JavaScript code is contained in a file, place the file in the following root location within your application server:

`IdentityMinder.ear\custom\validationscripts`

You can place the JavaScript file in a subdirectory of the root location—for example, `custom\validationscripts\myJavaScripts`.

Resource Bundles

We recommend that you deploy your custom resource bundles in the following location within the application server:

`IdentityMinder.ear\custom\resourceBundles`

If you choose not to store your custom resource bundles in `custom\resourceBundles`, you must store them in the custom directory or in any directory beneath it.

Email Templates

Email templates are deployed in subdirectories of the following location:

`IdentityMinder.ear\custom\emailTemplates`

Chapter 10: Task Execution Web Service

This section contains the following topics:

[What Are Web Services?](#) (see page 147)

[What Is TEWS?](#) (see page 148)

[Remote Task Requests](#) (see page 149)

[Task Operation Types](#) (see page 150)

[Web Service Configuration](#) (see page 152)

[Client Application Development](#) (see page 155)

[SOAP Messages](#) (see page 157)

[Exception Messages](#) (see page 158)

[Authentication](#) (see page 158)

[Authorization](#) (see page 159)

[Localization](#) (see page 162)

What Are Web Services?

Web services are self-describing modular applications that enable distributed computing over the Internet. Web services can provide anything from simple information requests to complicated business processes. Web services can be published, located, and invoked remotely across the Web.

Web service interfaces provide enough detail to allow the development of calling applications. This interface is typically described in an XML document called a Web Services Description Language (WSDL) file. Web service messages are defined using the SOAP protocol, and are conveyed using HTTP.

Software applications written in different programming languages and running on a variety of platforms can all use Web services to exchange data and processes. This interoperability (for example, between Microsoft Windows and Linux applications) is possible because Web service architecture is based on accepted open standards.

Web Service Protocols

The Web services protocols are used to define, locate, implement, and make Web services interact with each other. The following XML-based protocols specify the Web service interface as required by Identity Manager:

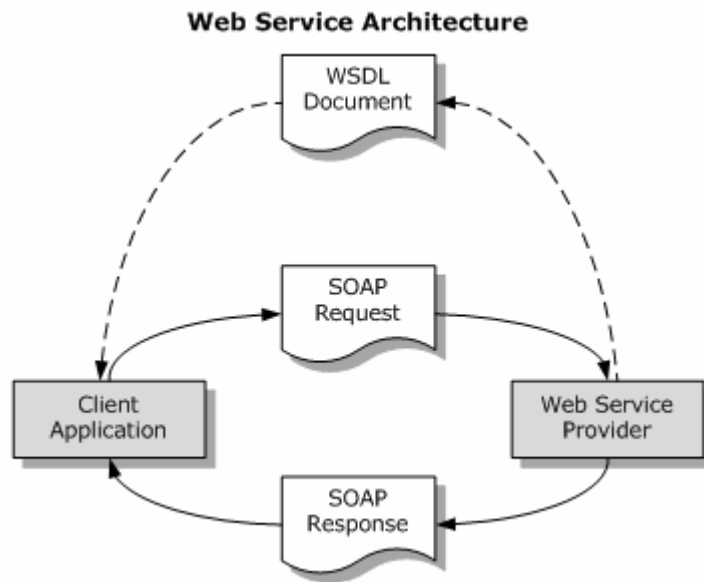
Simple Object Access Protocol (SOAP)

Defines the runtime message format that contains the service request and response.

Web Services Description Language (WSDL)

Describes the Web service interface, the SOAP message, and how the service is invoked.

A Web service can be thought of as a software service described in a WSDL document and formatted on the Web in a SOAP document. This relationship is illustrated in the following diagram:



What Is TEWS?

The Identity Manager Task Execution Web Service (TEWS) is a Web service interface that allows third-party client applications to submit remote tasks to Identity Manager for execution. This interface implements the open standards of WSDL and SOAP to provide remote access to Identity Manager. All Identity Manager tasks are supported.

A client application submits a remote task as an HTTP 1.x POST request to a Web service URL in the Identity Manager environment. The body of the POST request is a SOAP document that conforms to the interface described in a task-specific WSDL document generated by Identity Manager. The WSDL document describes the metadata that the client application needs to prepare and submit a task request.

Identity Manager replies to a remote task request with a SOAP response containing one of the following:

- The requested information
- Indication that the task has been submitted for processing
- An exception that identifies a problem with the request

The client application is responsible for gathering, formatting, and sending the information required for Identity Manager to perform the Web service task implementation. No Identity Manager APIs are provided for these purposes.

Remote Task Requests

Identity Manager processes a remote task request in the same way that it processes a task submitted on a task screen. For example:

- A remote request can be subject to the same validation checks and custom processing (such as business logic task handler and logical attribute handler processing) as a task submitted on a task screen.
- The task session contains the same information about a given task, regardless of the way the task request is submitted. The attributes that are available to a remote task request are the attributes that have been defined for the task during task screen configuration.
- Each remote task request is associated with a task-specific WSDL definition based on the task screen configuration.
- Remote task requests are submitted in the synchronous phase of task processing. The submitting application receives a response to a request before task processing begins, allowing the application to correct and resubmit the request if a validation error or other problem occurs.
- In the asynchronous phase of task processing, all Identity Manager functionality (for example, workflow processing, event listener processing, email generation, auditing) can be involved in the execution of a remotely submitted task.

Note: Identity Manager audits can be performed for execute operations, but not for search or query operations. Third-party products that provide Web service auditing are available.

More Information:

[Synchronous Phase](#) (see page 15)

[Asynchronous Phase](#) (see page 19)

How a Remote Request Is Processed

For example, when a client application submits a `ViewUserQuery` request to the Task Execution Web Service, the following actions occur:

1. The client application gathers the required information. In the case of `ViewUserQuery`, the required information consists of:
 - The task tag
 - The Identity Manager environment where the task is being submitted
 - The unique identifier of the administrator who is submitting the task
 - An identifier (such as a name or unique identifier) of the user whose profile information is being requested

This information can come from any source—for example, from user-supplied data on a custom screen, from a static configuration file-driven application, or from a live data feed from some external source.

2. The application formats the SOAP request according to the XML element `ViewUserQuery` in the WSDL definition.
3. The application sends the XML document to the Task Execution Web Service as an HTTP POST request.
4. When the Task Execution Web Service receives the request, it instructs Identity Manager to execute the operation and returns the status and profile information in a SOAP response.
5. Depending on the outcome of the request, the application processes the result (for example, displays the result in a custom UI) or takes an appropriate action based on any error information included in the result.

Task Operation Types

All Identity Manager tasks can be processed through the Task Execution Web Service. Tasks are associated with the following operational types:

Search

Retrieves a list of objects that match the search criteria. Typically, a search operation occurs at the beginning of a task. It retrieves the list of objects from which to select the target object (the subject) of the task.

This operation is similar to performing a search operation in the Identity Manager User Console.

Query

Requests profile information about a particular managed object, plus information about any of the object's relationships. For example, a query operation might retrieve a user's name, title, and email address, and also any of the user's roles and groups.

The specific attributes that are returned to the requesting application are the attributes that are defined on the tabs of the corresponding task screen.

This operation is similar to executing a View... task in the Identity Manager User Console.

Execute

Performs an Identity Manager action other than a view or self-view action—for example, creating, modifying, or deleting a user.

How Task Operations Work

Completing a typical Identity Manager admin task using remote requests is a three-step process that requires the three operation types (search, query, and execute) be called in sequence:

1. Retrieve a list of managed objects using the search operation which corresponds to the Identity Manager admin task.
2. Retrieve profile and relationship for a particular managed object using the which corresponds query operation.
3. Submit a remote request for Identity Manager to execute the admin action on the managed object.

Most admin task operations, such as `ModifyUser`, `ModifyGroup`, and `ModifyAdminTask`, have corresponding search and query operations for retrieving objects and profile data. For example, the three operations to modify a user are:

- `ModifyUserSearch`
- `ModifyUserQuery`
- `ModifyUser`

However, not all Identity Manager admin tasks require three steps calling different types of operations. The `SelfRegistration` operation is an example of a task that can be completed with one remote request.

A Task Operations Example

The following remote Web service calling sequence demonstrates the WSDL operations required to complete the Modify User task:

1. A search operation retrieves a list of objects that match the specified search criteria (for example, LastName=Smith).

The WSDL operation to call is ModifyUserSearch.

2. Programming logic in the client application selects a particular object from the list of returned objects (for example, the user object for John Smith). The selected object is the subject of the task.

3. A query operation retrieves profile and relationship information for the object selected in step 2. The query operation specifies the object to retrieve through information (such as a unique name) specified in the <Filter> or <Subject> element. Typically, the specified information includes information returned from the search operation.

The WSDL operation to call is ModifyUserQuery.

4. Once the object information is retrieved through the query, the information can be managed in the task.

The client application modifies one or more of the attribute values retrieved in the query.

5. The client application submits the remote request. If the request is submitted successfully, the task enters the asynchronous phase for task processing.

The WSDL operation to call is ModifyUser.

Web Service Configuration

Enabling Identity Manager tasks for remote Web services requires performing the following configuration steps:

- In the Management Console, Web services and WSDL generation must be enabled for the Identity Manager Environment.
- In the User Console, the individual task must be enabled for Web services.

Web Service Properties Screen

You configure Web services properties for the Identity Manager Environment in the Management Console.

To display the Web Services Properties screen:

1. Log into the Identity Manager Management Console.
2. Click Identity Manager Environments.
3. Click the name of the Environment where the Web services will be used.
4. Click Advanced Settings.
5. Click Web Services.

The Web Service Properties page appears.

Web Services Properties

There are four Web services properties:

Enable Execution

Check to enable the Task Execution Web Service (TEWS) for the environment.

By default, TEWS is disabled.

Note: If you enable TEWS for the environment, you must restart the application server.

Enable WSDL Generation

Check to enable WSDL generation for the environment.

WSDL generation is enabled by default. The URL to perform WSDL generation is not accessible when WSDL generation is disabled.

Note: When generating WSDLs in a clustered environment, you should run one Policy Server and one Identity Manager Server, and stop all other Policy Servers and Identity Manager Servers within the cluster.

Enable admin_id (allow impersonation)

Specifies whether TEWS supports impersonation.

When this option is selected, TEWS uses the admin ID found in the SOAP message sent to the Web service to authenticate the request.

When this option is not selected, the ID of the user who generated the request is used to authenticate the request.

SiteMinder Authentication

When Identity Manager is integrated with SiteMinder, you can configure SiteMinder to secure the URL for Web services.

- **None (default)**

SiteMinder does not secure the Web services URL. In this case, you should use an external method to secure the Web services URL.

- **Basic Authentication**

SiteMinder basic authentication secures access to the Web services URL.

When selected, Identity Manager automatically creates the authentication schemes, domain, realm, rule, and policies needed to secure the URL for Web services. You can modify these configuration settings in the SiteMinder Policy Server user interface.

- **Other**

SiteMinder is used to authenticate Web service requests, but Identity Manager does not create the required policy objects. When this option is selected, users configure the policy objects in the SiteMinder Policy Server user interface.

Note: For information on issuing remote requests to Identity Manager through the Task Execution Web Service, see the *Developer's Reference*. For more information about using SiteMinder to protect a Web services URL, see the *CA SiteMinder Web Access Manager Policy Server Configuration Guide*.

More information:

[WSDL Generation](#) (see page 156)

Enable Web Services for Individual Tasks

In the User Console, you need to verify that the tasks you want to invoke are enabled for Web services.

To enable Web services for tasks:

1. Log into the Identity Manager User Console.
2. Select the Roles and Tasks tab.
3. Select Admin Tasks.
4. Select Modify Admin Task.
5. Search for and select the admin task (for example, Create User).

6. Check Enable Web Services.

Note: This setting is checked by default.

7. Click Submit to save your changes.

Note: For instructions about configuring tasks, see the *CA Identity Manager Administration Guide*.

Client Application Development

This is an overview of the client application development process for submitting remote Web service requests to Identity Manager.

Note: This overview assumes that your IDE includes a third-party tool like Axis that generates proxies. While this approach automates some of the development, it is not required.

1. Make sure that Web services are enabled for the Identity Manager Environment and for each task that is to support remote requests.
2. Generate the WSDL code by invoking the URL:

`http://<host:port>/idm/TEWS6/<environmentalias>?wsdl`

For example:

`http://tewstest.ca.com:7001/idm/TEWS6/neteauto?wsdl`

Note: The URL is case-sensitive, and <environmentalias> is the protected alias of the Identity Manager Environment.

3. In your integrated development environment (IDE), create the proxies based on the WSDL document generated in the preceding step.

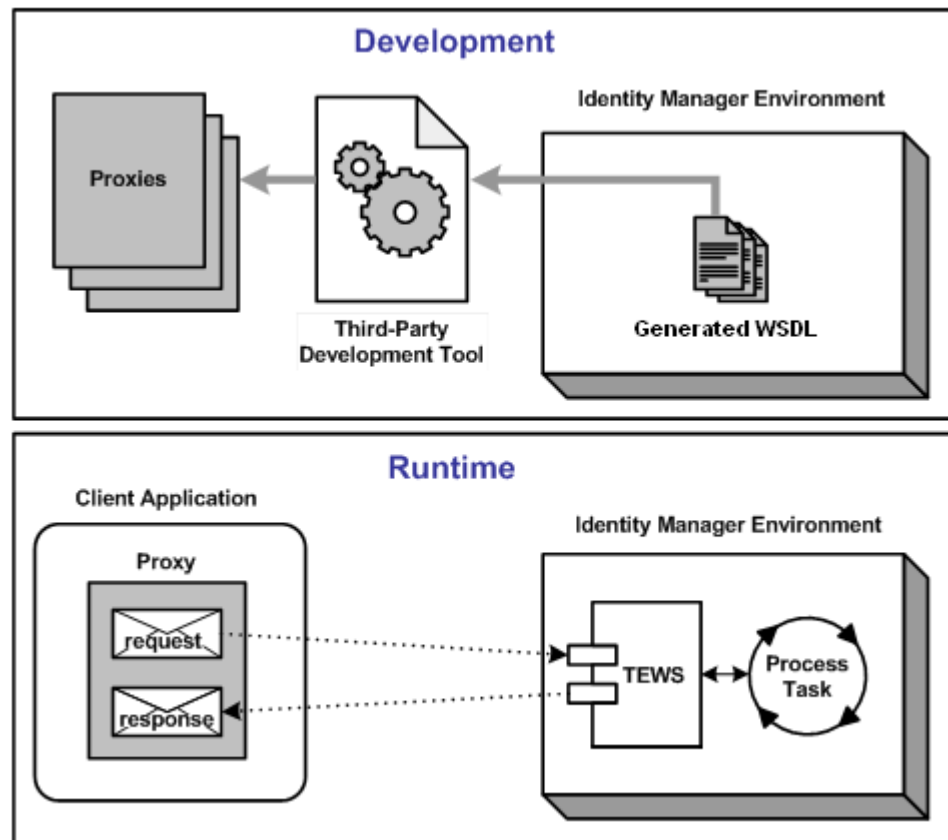
Proxies reflect the operations that your client can submit to Identity Manager. Java proxies generated by third-party tools such as Apache Axis provide precise, self-documenting entry points and parameters.

4. In your IDE, develop the client code that will formulate and submit the remote requests and process the responses. Typically, you develop this code using a third-party tool such as Apache Axis and Microsoft .NET .

Develop whatever programming logic is needed to process the SOAP requests and responses in compliance with your business requirements.

5. At run-time, the client application submits an Identity Manager request through a generated proxy. The proxy transforms the request into a SOAP document that the Task Execution Web Service can understand, and then submits the request to the Web service.
6. Identity Manager processes the request and returns a SOAP response to the client application.

The following diagram illustrates the development and run-time processes:



WSDL Generation

You request a WSDL document from Identity Manager by invoking the following URL:

```
http://<host:port>/idm/TEWS6/<environmentalias>?wsdl
```

For example:

```
http://tewstest.ca.com:7001/idm/TEWS6/neteauto?wsdl
```

The URL is case-sensitive, and **<environmentalias>** is the protected alias of the Identity Manager Environment.

If WSDL generation is enabled for the Identity Manager Environment, the response from Identity Manager is a single generated WSDL document. You can use this WSDL document directly to generate proxy code with a development tool like Axis WSDL2Java.

The WSDL document contains a class definition for each task that is enabled for Web services in the Identity Manager User Console. By default, all tasks are enabled, so the WSDL document is quite large.

A development IDE like Eclipse Web Tools Platform (WTP) enables you invoke the URL and display the WSDL file along with a list of all possible operations. You can also supply operation input values, execute the operation, and display the SOAP request and response messages.

Eclipse WTP, IntelliJ, and .NET include plug-ins that test Web services by parsing the returned WSDL and rendering a test interface to call the available services with no manual coding required.

If WSDL generation is disabled, one of the following HTTP errors is returned with additional message information:

- 404 file not found
- 500 not authorized

Note: When generating WSDLs in a clustered environment, you should run one Policy Server and one Identity Manager Server, and stop all other Policy Servers and Identity Manager Servers within the cluster.

More Information:

[Web Service Properties Screen](#) (see page 153)

SOAP Messages

The Task Execution Web Service expects messages from the client to have the following characteristics:

- Message exchanges between the client and the Web service are in the form of a SOAP Request-Response message exchange pattern (MEP).
- The ultimateReceiver for all requests is the Web service.
- The Web service throws a fault for any MustUnderstand SOAP header block. Also, NotUnderstood header blocks are generated for headers that cause this fault.
- A binding on a Web service-enabled operation is a SOAP binding in the document binding style.

Exception Messages

If an error occurs during the processing of a remote request, Identity Manager reports the error to the client in standard SOAP fault format. For example:

- The <Value> sub-element contains a SOAP fault code.
- The <Reason> sub-element contains a readable description of the fault.
- The <Detail> sub-element contains Web service-specific information about the exception. The information is provided in the <ImsException> element of the ImsStatus.xsd schema, as follows:
 - The <name> sub-element contains the name of the Web service exception.
 - The <code> sub-element contains a Web service-specific exception code. The exception codes allow the client application to include error-handling logic.
 - The <description> sub-element contains a human-readable description of the Web service exception.

In accordance with SOAP standards, HTTP return codes are set as required.

Note: If an exception occurs with a remote request, Identity Manager reports the exception to the client. In accordance with SOAP standards, no further processing is performed on the request.

More Information:

[TEWS Error Codes](#) (see page 173)

Authentication

The URL for generating a WSDL is typically protected by a CA SiteMinder Agent. If you need a more stringent Task Execution Web Service security model, consider using SOA Security Manager to secure it. SOA Security Manager is available separately from CA.

SiteMinder and Security Manager authentication schemes, such as SiteMinder Basic and Certificate-based schemes, can be used to protect the URLs. Because each end point is associated with a specific operation and a unique URL, you can assign different authorization schemes and protection levels to operations that have different security requirements.

Authentication challenges must be presented as part of the authentication protocol if they are not included in the HTTP POST request. If challenges are presented as part of the authentication protocol, user credentials must be provided programmatically.

When SiteMinder or Security Manager authenticates the user credentials that the client application presents to it, details about the session are inserted into the header of the response. This information includes the ID of the administrator who is executing the task.

The session information remains in the HTTP POST header. Identity Manager may obtain the admin ID from this session information when it determines whether the administrator is authorized to perform the task.

More Information:

[Web Service Properties Screen](#) (see page 153)

Authorization

Whether the submitted task can be executed depends upon the rights of the administrator who is making the request. The administrator's ID is specified in one of two ways:

- As part of the session information that SiteMinder or TransactionMinder inserts into the POST request header during the authentication response. This is the default.
- Through the <admin_id> element that is passed to Identity Manager in the body of the POST request.

Using the admin ID provided in the session header guarantees that the administrator who is issuing the remote request has been authorized to do so by SiteMinder or TransactionMinder. However, no authorization checks are performed on an admin ID supplied through the <admin_id> element of the request.

If no session information is available in the header and no admin ID is supplied in the <admin_id> element, Identity Manager returns a Not Authorized exception in the HTTP POST response body.

Authorization is not required for a task configured as a Public Task.

Administrator Specification

By default, Identity Manager uses the ID of the administrator that SiteMinder or SOA Security Manager has already authorized and whose ID is included in the session information.

You can change this default by modifying the web.xml file. The location depends on your environment:

- For JBoss and WebLogic:
IdentityMinder.ear\user_console_war\WEB-INF
- For WebSphere:
WebSphere-ear\IdentityMinder.ear\user_console_war\WEB-INF

Whether Identity Manager uses the ID specified in the session or through an `<admin_id>` element is determined by the values of the `<param-value>` elements in the following default section of web.xml:

```
<servlet>
  <servlet-name>Tews6Servlet</servlet-name>
  <display-name>IM 6.0 Task Execution WebService Servlet</display-name>
  <servlet-class>
    com.netegrity.ims.tews.Tews6Servlet
  </servlet-class>
  <init-param>
    <param-name>use_admin_id</param-name>
    <param-value>>false</param-value>
  </init-param>
  <init-param>
    <param-name>require_sm_headers</param-name>
    <param-value>>true</param-value>
  </init-param>
</servlet>
```

Administrator Parameters

use_admin_id = false
require_sm_headers = true

The admin ID in the session. The ID has been authorized for this request by SiteMinder or SOA Security Manager.

These are the default settings.

With these settings, Identity Manager uses the ID in the session, whether or not an ID is specified in `<admin_id>`.

Code Example

```
<init-param>
  <param-name>use_admin_id</param-name>
  <param-value>false</param-value>
</init-param>
<init-param>
  <param-name>require_sm_headers</param-name>
  <param-value>true</param-value>
</init-param>
```

use_admin_id = true
require_sm_headers = false

The admin ID specified in <admin_id>. Identity Manager does not check whether the administrator is authorized for the request.

With these settings, Identity Manager does not check the session information for an ID.

Typically, these settings are used if SiteMinder or Security Manager is not protecting the Web service end point.

Code Example

```
<init-param>
  <param-name>use_admin_id</param-name>
  <param-value>true</param-value>
</init-param>
<init-param>
  <param-name>require_sm_headers</param-name>
  <param-value>>false</param-value>
</init-param>
```

use_admin_id = true
require_sm_headers = true

The admin ID specified in <admin_id>. Identity Manager does not check whether the administrator is authorized for the request.

With these settings, Identity Manager validates that SiteMinder headers are present in the request.

Typically, these settings are used to provide impersonation functionality for the request.

Code Example

```
<init-param>
  <param-name>use_admin_id</param-name>
  <param-value>true</param-value>
</init-param>
<init-param>
  <param-name>require_sm_headers</param-name>
  <param-value>true</param-value>
</init-param>
```

The Channel Connection

There are two segments in the channel from the client application to the Task Execution Web Service:

- From the client to the Web Server
- From the J2EE application server Web Server proxy to the J2EE server

You should consider using an SSL connection to protect the clear-text SOAP documents sent over this channel. You can set up most J2EE application server proxies to use an SSL connection.

Localization

When a client sends a remote task request to Identity Manager on the server, the client can specify the language that Identity Manager should use for all of its communications with the client:

- If the client request specifies one or more language preferences in the HTTP ACCEPT_LANGUAGE request header, the first supported language in the list is used.

For example, the server receives the following ACCEPT_LANGUAGE header value:

```
fr, ja, en-us, hi
```

The first (leftmost) language preference determines the client's locale and the language in which Identity Manager will communicate with the client—in this case, French. If Identity Manager cannot find a resource bundle that corresponds to the first language preference in the list, the next preference is used. If Identity Manager cannot find a resource bundle for any of the preferences in the list, the default locale and language are used.

- If the client request does not specify a language preference in the `ACCEPT_LANGUAGE` request header, the language associated with the client's default locale is used.

Note: For more information about localization, see the *CA Identity Manager Configuration Guide*.

Chapter 11: TEWS Sample Client Code

This section contains the following topics:

[Web Service Development Workstation](#) (see page 165)

[Third-Party Software](#) (see page 165)

[Web Service Samples](#) (see page 166)

[Java Samples for Axis](#) (see page 167)

[Build Script for Axis](#) (see page 168)

[Running the Java Classes](#) (see page 168)

Web Service Development Workstation

Your Identity Manager Web service development workstation should include installations of the following third-party software:

- Apache Ant 1.5 (or later)
- Apache Axis 1.3 (or later)
- Eclipse WTP (or other IDE)
- Sun J2SDK/J2RE 1.4.2_7

On the development machine, you also should create a Java project directory that contains the Web service sample files installed with Identity Manager.

The development machine does not need to be the same machine where Identity Manager is installed.

Note: Because Eclipse requires significant memory to work, we recommend a PC workstation with at least 1 GB RAM.

Third-Party Software

A Web service development environment should include installations of the following third-party software:

Apache Ant 1.5 (or later)

Apache Ant is a Java-based utility for automating the building of Java projects. It is similar to make, however Ant uses XML to describe the build process whereas make has its own Makefile format. By default, the Ant XML file is named build.xml.

<http://ant.apache.org>

Apache Axis 1.3 (or later)

Apache Axis is an implementation of the Simple Object Access Protocol (SOAP) server. It includes various utilities and APIs for generating and deploying Web service applications. Using Apache Axis, developers can create distributed Web service applications.

<http://ws.apache.org/axis/>

Eclipse WTP

Eclipse is a widely used Java IDE (Integrated Development Environment). Instead of using the standard Eclipse Java IDE package, we suggest that you download the Eclipse Web Tools Platform (WTP) package, which extends the standard Eclipse Java IDE with tools for Web and Java EE applications, including Web services.

<http://www.eclipse.org/webtools/>

Note: Eclipse WTP is recommended but not required.

Sun J2SDK/J2RE 1.4.2_7

The Sun Java 2 Software Development Kit (SDK) and Run-time Engine (RTE) are required for Identity Manager Java development. As of this writing, the CA supported version is Sun J2SDK/J2RE 1.4.2_7, which you can download from the Sun archive.

<http://java.sun.com/products/archive/>

Web Service Samples

Identity Manager ships with a number of samples for common use cases. The samples are located in the following Identity Manager installation directory:

`samples\WebService`

- The Axis sub-directory contains Java client samples that use proxies generated by Apache Axis.
- The .NET sub-directory contains C# client samples that demonstrate how to invoke Identity Manager tasks via Web services.
- The XML sub-directory contains samples of task requests and responses formatted as SOAP documents.

There are also Readme.txt files, which contains prerequisites and instructions for using the samples.

Java Samples for Axis

The Java samples for Axis demonstrate how to invoke Identity Manager tasks using remote Web service calls. The Java samples use the open source Axis Java framework and Axis WSDL2Java utility to generate Java proxies from the Identity Manager WSDL file.

The samples are a set of Java classes for performing basic administrative tasks, for example:

- Creating a new user
- Changing a password
- Modifying user data

There is also a directory of optional sample classes that require additional configuration of the NeteAuto Environment. For example, the following samples require workflow and a provisioning directory:

- Approving a work item
- Creating a provisioning role
- Modifying a provisioning role

Java Sample Classes

The following list provides brief descriptions of a subset of the Java sample classes:

CreateUser

Logs in as SuperAdmin and creates a new Identity Manager user named New Supplier with a minimal set of user attribute values.

ChangeMyPassword

Logs in as New Supplier and changes the original default password.

ModifyMyProfile

Logs in as New Supplier and changes address-related attribute values.

ModifyUser

Logs in as SuperAdmin and changes employee-related attribute values.

UserSearch

Logs in as SuperAdmin and searches for all users by matching the asterisk (*) wildcard character.

ViewMyRoles

Logs in as New Supplier and returns roles for that user.

ViewOrg

Logs in as SuperAdmin and views the organization named USA.

ViewUser

Logs in as SuperAdmin and views all users with first names matching the string "New".

ForgottenPassword

Logs in as the user identified by the %USER_ID% wellknown attribute to request a new temporary password.

DeleteUser

Logs in as SuperAdmin and deletes the user named New Supplier.

Build Script for Axis

The Axis samples directory includes the following Apache Ant build script that generates Java proxies from WSDL, compiles the Java samples, and then runs the compiled samples:

`samples\WebService\Axis\build.xml`

The following file provides instructions about how to configure and run the script:

`samples\WebService\Axis\Readme.txt`

Running the Java Classes

This is an overview of the process for building and running the Java samples for Axis:

1. Verify that your workstation meets the development requirements.
2. In the Management Console, enable Web services and WSDL Generation.
3. In the User Console, verify that the tasks you want to invoke are enabled for Web services.
4. Edit the build.xml script to reflect your development environment.
5. Run build.xml to build the WSDL proxies and compile the Java samples.
6. Run build.xml again to execute the sample classes.

Note: See the following file for more detailed instructions:

`samples\WebService\Axis\Readme.txt`

Appendix A: TEWS Client Migration

This section contains the following topics:

[Migrating to Identity Manager r8.1](#) (see page 169)

[Migrating to Identity Minder r6.0](#) (see page 171)

Migrating to Identity Manager r8.1

The following sections apply when migrating to Identity Manager r8.1 from an earlier version of the product.

Namespace and Package Name Changes

Starting with Identity Manager r8.1, WSDL generation is invoked using a specific URL that generates a single WSDL file with a single namespace, without references to any other files. Any Web service client code written for earlier versions of Identity Manager TEWS may need to be modified.

The single namespace requires the removal of all the namespaces that previously had to be defined to import a WSDL file for each task. This results in a major change when generating the proxies and reduces multiple generated package names to one package name. Your client code will need to be modified to reflect changed package names.

Attribute Name Changes

In Identity Manager r8.1, Axis 1.2 beta was no longer supported. This change results in enhanced performance, and it affects client code in that the generated proxies vary with the different Axis versions. For example, underscore characters used in attribute names have been eliminated, resulting in more readable code.

Result and Status Proxy Values

Prior to Identity Manager r8.1, if a service returned both a result and a status, special holder classes were generated for proxies so the result and status could be returned by reference from the method call. This exposed a parsing problem with Axis, and required a special patched version of Axis to be shipped with Identity Manager.

In Identity Manager r8.1, the result and status were returned as a single value rather than as two values by reference. This eliminated the requirement to use the patched version of Axis to generate proxies.

Result and Status Code

Result and status information are contained within a single return value which simplifies client code. The following example demonstrates the holder classes used in previous Identity Manager releases:

```
/* output parameter for status part of the message */
_ImStatusHolder statusHolder = new _ImStatusHolder();

/* output parameter for the return message part */
_ViewMyRolesQueryResultHolder resultHolder =
    new _ViewMyRolesQueryResultHolder();

/* this forms the SOAP request, and processes the return document */
port.viewMyRolesQuery(admin_id,vmrq,statusHolder,resultHolder);
```

In Identity Manager r8.1, the same code returns a value without the need for a holder class. The status can now be queried directly:

```
ViewMyRolesQueryResult result = port.viewMyRolesQuery(admin_id,vmrq);
ImStatus imsStatus = result.getImStatus();
```

Public and Protected Context Tasks

All context information for TEWS tasks has been encapsulated into public and protected context types. These types are defined in the WSDL and are the first parameter for all tasks.

Prior to Identity Manager r8.1, all tasks required the `admin_id` as part of the request message, regardless of whether they were public or protected. Now, the WSDL does not define an `admin_id` attribute in request messages for public tasks. Therefore, `admin_id` is not required for public tasks, although it is still required for protected tasks. This may impact migrating customer code that currently sets the `admin_id` for public tasks.

In addition, each context type lists a number of optional attributes that apply to particular tasks. For example, the protected context includes `workitem_id` and `action` attributes which are expected for approval tasks.

Context Code Changes

The execution of a task in TEWS includes code that evaluates the information supplied in the context. Values that are required for a given task are checked, and any appropriate SOAP fault is generated.

If attributes that are not required are specified anyway, warnings are logged on the console and the values are ignored. This change allows for the future addition of context attributes without impacting the proxies that are generated. Therefore, context changes require existing client code to be refactored to account for the context type that is now included in rebuilt proxies.

The following code example does not consider context:

```
EnableDisableUserSearchResult result =  
    searchPort.enableDisableUserSearch(strAdminDN, es);
```

Whereas migrated code must take the context into account:

```
TaskContext ctx = new TaskContext();  
ctx.setAdmin_id(strAdminDN);  
EnableDisableUserSearchResult result =  
    searchPort.enableDisableUserSearch(ctx, es);
```

Locators and Ports

In Identity Manager r8.1, the number of ports changed in the generated WSDL. In previous releases, a separate port was defined for each task. With the single WSDL, there is one port defined for the protected alias and one port defined for the public alias. Tasks are sorted into these two ports based on whether the task is public or protected. This means that with most proxies, the location of the Web service can be changed with a single method call.

In general, calling an operation now uses code similar to the following example:

```
Tews6Locator locator = new Tews6Locator();  
Tews6PortType port = locator.getPort();  
/* access operations on port */
```

The same port can be used for multiple operations. Previous releases required code similar to the following example, in which each task had its own port:

```
Tews6Locator locator = new Tews6Locator();  
ChangeMyPasswordPortType port = locator.getChangeMyPasswordPortType();
```

Migrating to Identity Minder r6.0

The following section applies when migrating to Identity Minder v6.0 from an earlier version of the product.

Disable Prior Versions

The Task Execution Web Service underwent a major revision for Identity Minder v6.0. Prior versions are deprecated.

Identity Minder distinguishes between remote requests issued to the current version of the Web service and to a deprecated version. Identity Minder processes each request appropriately. However, you may prefer to disable the deprecated version.

To disable a prior version of the Web service

1. Open the file web.xml.
2. Comment out or delete the following entries:

```
<filter>
  <filter-name>SOAPRemoverFilter</filter-name>
  <filter-class>
    com.netegrity.ims.tews.TewsSOAPRemoverFilter
  </filter-class>
</filter>

<filter-mapping>
  <filter-name>SOAPRemoverFilter</filter-name>
  <url-pattern>/TEWS</url-pattern>
</filter-mapping>

<servlet>
  <servlet-name>TewsServlet</servlet-name>
  <display-name>
    Task Execution WebService Servlet
  </display-name>
  <servlet-class>
    com.netegrity.ims.tews.TewsServlet
  </servlet-class>
</servlet>

<servlet-mapping>
  <servlet-name>TewsServlet</servlet-name>
  <url-pattern>/TEWS</url-pattern>
</servlet-mapping>
```

Appendix B: TEWS Error Codes

Identity Manager reports Web service-specific exception information in the <ImException> element of the SOAP response. The <ImException> information is presented in the <Detail> element of a SOAP fault.

The following table lists the Identity Manager exception codes and their meanings:

Code	Fault Code	Description
200	Receiver	Web service is not available. If the Web service is not enabled for the environment, the request fails without any further processing.
300	Sender	Error parsing the request.
400	Sender	Authorization failure. This error occurs in either of these cases: No task session exists. The <admin_id> element of the request is empty, and Identity Manager cannot retrieve the administrator's ID from the task session.
500	Sender	Task session error. Each task session error is reported individually.
700	Receiver	General error that does not apply to one of the above categories.

Note: There is no Identity Manager detail information associated with the SOAP fault code MustUnderstand. This error occurs when one or more mandatory SOAP header blocks are not understood. NotUnderstood header blocks are generated for headers that cause this fault.

Task Session Error Codes

This table lists the task failure error codes returned by Identity Manager.

Error Code	Name	Comments
502	Task not found	Unable to create a task session and execute the task because the specified task name was not found in the IME.
503	IME not found	Unable to find an IdentityMinder environment with the specified name.
504	Operand not found	The managed object specified as a parameter of the task to be executed is not found. For example, when executing the "Modify User" task, the specified user is not found in the user store.
505	Task parameter not found	The name of the task parameter is incorrect for the specified task.
507	Bad task parameter data	Three conditions can cause this error message: one of the values specified in a task parameter is invalid; the administrator has no privilege on this object; the operation processing the parameter may have failed.
510	Unknown error	Unknown error
511	Admin not found	The specified administrator was not found.

Index

A

- ACCEPT_LANGUAGE request • 166
- access
 - events • 124
 - managed objects • 114
- access tasks
 - in managed objects • 110
 - reserved fields • 144
- AccessRole interface • 110
- AccessRoleEvent interface • 126
- AccessTask interface • 110
- AccessTaskEvent interface • 126
- ActionType class • 144
- adapter base class
 - Business Logic Task Handler API • 67
 - Logical Attribute API • 41
 - Notification Rule API • 105
 - Participant Resolver API • 99
- adaptor objects • 27
- addMessageObject() • 73
- addUserMessage() • 138
- ADMIN notification rule • 103
- administrator
 - eTrust SiteMinder • 164
 - eTrust TransactionMinder • 164
 - ID • 164
 - policy • 141
 - roles • 110
 - rules • 141
 - tasks • 15, 109, 110
- AdminRole interface • 110
- AdminRoleEvent interface • 126
- AdminTask interface • 110
- AdminTaskEvent interface • 126
- after() • 81, 82, 85
- Apache Axis • 171, 173
- API
 - adaptor objects • 27
 - asynchronous phase model • 26
 - Business Logic Task Handler • 67
 - context objects • 27
 - Email Template • 104
 - examples of use • 14, 23
 - functional model components • 27
 - functional model overview • 25
 - in asynchronous operations • 20
 - in synchronous operations • 16
 - Logical Attribute • 41
 - Notification Rule • 105
 - overview • 12
 - Participant Resolver • 99
 - purpose • 109
 - synchronous phase model • 25
 - Workflow • 94
- ApiResultSet class • 143
- approval task • 99
- approvals
 - for generated events • 122
 - workflow • 12
- approved() • 81, 85
- APPROVERS_REQUIRED • 99
- AssignableObject interface • 112
- assignResources() • 66
- asynchronous phase • 15, 19, 20, 26
- Attribute Name dropdown list • 31, 56
- attribute validation
 - about • 145
 - business logic task handlers • 123
 - directory-level • 145
 - exceptions • 137
 - JavaScript • 39, 63
 - objects used in • 145
 - regular expression • 39, 63
 - task data • 66
 - task-level • 145
 - user input • 40, 43
- AttributeChangeType class • 144
- AttributeCollection interface • 111, 113, 119
- AttributePermissionException • 119
- AttributeRight class • 119, 140
- AttributeRightsCollection class • 119, 140
- attributes
 - about • 31
 - committing changes • 114
 - expressions • 143
 - including in a retrieved object • 119
 - logical and physical • 32, 33, 113
 - managed object • 12, 113, 114
 - mapping, in directory.xml • 34
 - permissions • 119, 140
 - provider access • 119

- See also logical attributes, physical attributes, w • 31
- task context access • 123
- AttributeValidationException class • 145
- AttributeValidator class • 145
- AuditEvent interface • 127
- auditing
 - changes to be included • 144
 - events • 122, 127
 - unavailable with providers • 117
- Axis client • 171, 173

B

- base resource bundle • 138
 - event descriptions • 128
 - IMSEException • 137
 - IMSResources • 128, 140
- BASE search depth type • 144
- before() • 81, 84
- BLTHAdapter class • 67
- BLTHContext interface • 64, 67
 - managed object access • 114
- Business Logic Task Handler API
 - access to managed objects • 114
 - BLTH. See Business Logic Task Handler API,
 - business logic task • 63
 - called by Identity Manager • 63
 - calling sequence • 72
 - compiling • 149
 - data persistence • 67
 - handleSetSubject() • 16, 23
 - handleStart() • 16, 23
 - handleSubmission() • 16, 23
 - handleValidation() • 16, 23
 - objects • 67
 - sample • 69
 - task example • 23
 - use cases • 66
- business logic task handlers
 - alternative validation methods • 63
 - attribute validation • 123
 - configuring • 77
 - executed after logical attribute • 43, 72
 - global • 66
 - JavaScript implementation • 75
 - managed object access • 70
 - order of execution • 72
 - parallels with task screen operat • 121
 - regular expression validation • 63

- sample • 69
- scope • 66
- sharing data with other objects • 67
- task session • 64, 121
- task-specific • 66
- validation • 123
- business policies. See identity policies • 136

C

- calling sequence
 - Business Logic Task Handler API • 72
 - Logical Attribute API • 43
 - Participant Resolver API • 101
- calls to
 - Business Logic Task Handler API • 63
 - Logical Attribute API • 39, 43
 - Notification Rule API • 103
- casting events • 125, 136
- committing changes to attributes • 110, 114
- compiling source files • 149
- configuring
 - business logic task handlers • 77
 - forgotten password handler • 54
 - logical attributes • 45
 - notification rules • 107
 - option lists • 46, 48
 - participant resolvers • 102
- Confirm Password Handler • 35
- ConjunctionType class • 144
- constraints for searches • 143
- contacting technical support • iii
- context objects • 27
- Custom type participant resolver • 97
 - order of precedence • 101
- customer support
 - customer support, contacting • iii
- customizing Identity Manager
 - Business Logic Task Handler API • 23, 67
 - examples • 14, 23
 - localization • 139
 - Logical Attribute API • 23, 32, 41
 - Notification Rule API • 23, 105
 - Participant Resolver API • 23, 98, 99
 - Workflow API • 23, 94

D

- data flow • 32
- data persistence
 - Business Logic Task Handler API • 67

- Logical Attribute API • 41
- Workflow API • 94
- data store • 31
 - direct access through providers • 117
 - when data is committed • 114, 122
- decrypt() • 53
- deploying files • 150
- directory.xml • 34
- DirectoryProvidersCollection interface • 145
- DisabledReasonType class • 144
- dotNET sample files • 171
- Drop Down Menu • 47

E

- EAR file • 150
- email notifications • 104
- Email Template API
 - about • 104
 - and compilation • 149
 - task example • 23
- email templates
 - automatic generation of email • 104
 - deploying • 150
 - recipient list • 103
- enable logical attribute • 35
- Enable User Handler. • 35
- %ENABLED_STATE% • 35
- encrypt() • 53
- EnhancedAuditData class • 127
- eTrust Admin provisioning • 109, 110, 142
- eTrust IdentityMinder • 178
- eTrust SiteMinder • 162, 163, 164
- eTrust TransactionMinder • 162, 163, 164
- Event Listener API
 - access to managed objects • 114
 - compiling • 149
 - components • 83
 - EventContext • 83
 - EventListenerAdaptor • 83
 - sample code • 87
 - summary • 81
 - task example • 23
 - use cases • 82
- event listeners
 - and managed objects • 88
 - and tasks • 79
 - configuration • 89
 - event generation • 89
 - execution • 84, 87
 - invocation rules • 87
 - sample code • 87
 - states • 80, 85
- EventContext interface • 83, 84
 - API model components • 27, 83, 84
 - asynchronous phase model • 26
 - generateEvent() • 88
 - managed object access • 88, 114
 - read/write access • 124
 - synchronous phase model • 25
 - Workflow API • 94
- EventListenerAdaptor • 83, 84, 85
- EventProfile class • 127
- EventROContext interface
 - API model components • 27
 - asynchronous phase model • 26
 - managed object access • 114
 - Notification Rule API • 105
 - Participant Resolver API • 99
 - read/write access • 124
 - synchronous phase model • 25
- events
 - access to • 124
 - accessing managed objects through • 125, 126
 - AddToGroupEvent • 16, 79, 89
 - AssignAccessRoleEvent • 79
 - AssignAdminRoleEvent • 82
 - auditing • 122, 127
 - base interfaces for • 126, 128
 - casting a particular event • 125, 136
 - categories • 128
 - CreateUserEvent • 16, 79, 82
 - generating • 94, 114, 124
 - in asynchronous operations • 20
 - in asynchronous phase model • 26
 - in synchronous operations • 16
 - in synchronous phase model • 25
 - in tasks • 79
 - localizing event descriptions • 128
 - ModifyUserEvent • 82, 89
 - names • 125, 137
 - not generated with providers • 117
 - objects • 124, 125, 128
 - overview • 16
 - RemoveFromGroupEvent • 88
 - when generated • 122
- exceptions
 - displaying • 74

- IMSEException class • 137
 - remote request • 162, 182
 - storing • 138
- ExposedEventContextInformation • 104, 124
- ExposedTaskContextInformation • 104
- external tasks • 126
- extractAuditData() • 127

F

- Field properties box • 56
- fields
 - option list • 47
 - reserved • 144
- Filter type participant resolver • 97
 - order of precedence • 101
- filters for searches • 143
- Forced Password Reset Handler • 35
- forcePasswordReset logical attribute • 35
- Forgotten Password Handler • 35, 53, 54
 - customizing • 53
- Forgotten Password task • 53

G

- generateEvent() • 88
- generating secondary events • 94, 114, 124
- GenericAuditEvent class • 127
- getAdminTask() • 122
- getDisabledUsers() • 122
- getEnabledUsers() • 122
- getEvent() • 125
- getEventName() • 125
- getGroupMembers() • 122
- getNotifiers() • 20, 103
- getOrganization() • 50
- getQuestions() • 53
- getUser() • 122
- getUserLocale() • 138
- global business logic task handlers
 - about • 66
 - configuring • 77
 - order of execution • 72
- Grantor interface • 112
- group events • 128
- Group interface • 110
- GROUP object type • 35
- Group type participant resolver • 97
 - order of precedence • 101
- GroupableObject interface • 112
- GroupEvent interface • 126

- groups
 - constraints • 143
 - managed object • 110
 - membership • 112
 - subject's relationships • 121
 - tabs and • 121
- GroupSubscription logical attribute • 35

H

- handlers. See business logic task handlers,
logical attribute h • 32
- handleSetSubject JavaScript method • 75
- handleSetSubject() • 23, 66, 72
- handleStart JavaScript method • 75
- handleStart() • 16, 23, 72
- handleSubmission JavaScript method • 75
- handleSubmission() • 16, 23, 72
- handleValidation JavaScript method • 75
- handleValidation() • 16, 23, 72

I

- IAttributeValidator interface • 145
- IAttributeValidator.StringRef class • 145
- identifier for organizations • 98
- Identity Manager APIs
 - Business Logic Task Handler API • 67
 - Email Template API • 104
 - Logical Attribute API • 41
 - Notification Rule API • 105
 - Participant Resolver API • 99
 - Workflow API • 94
- Identity Manager Directory • 109
- identity policies • 136
- IdentityMinder • 178
- IdentityMinder.ear • 150
- IM Approvers • 101
- IMContext interface • 25, 26, 27, 83, 84
- IMEvent interface • 125, 136
- IMEventName interface • 137
- IMPersistentContext interface • 83
 - API model components • 27
 - asynchronous phase model • 26
 - Business Logic Task Handler API • 67
 - Event Listener API • 84
 - Logical Attribute API • 41
 - Logical Attribute Handler API • 41
 - synchronous phase model • 25
 - Workflow API • 94
- ims.jar • 149

- imsapi6.jar • 149
- IMSEException class • 137, 182
- IMSEExceptions.properties • 137
- ImsManagedObjectAttr element • 34
- IMSResources bundles • 128, 140
- IMSStatus • 154
- initialize()
 - default logical attribute values • 43
- initializeOptionList() • 43, 46
- integration with third-party applications • 96
- invalid passwords • 140, 144
- ISO 639 language identifiers • 138

J

- JAR files • 149
- JavaScript
 - attribute validation • 39, 63
 - business logic task handlers • 16, 75
 - deploying files • 150
 - email notification • 23
 - field initialization • 16
 - field validation • 16
- JSP file • 140

L

- language identifiers, ISO 639 • 138
- locale
 - resource bundles • 139, 140
 - retrieving for current user • 138
- localization
 - ACCEPT_LANGUAGE request • 166
 - event descriptions • 128
 - exception messages • 74, 137
 - extracting localized text • 140
 - identifiers • 138, 140
 - IMSEExceptions.properties • 137
 - IMSResources bundle • 140
 - new skins • 140
 - strings • 139
- Localizer class • 139
- Logical Attribute API
 - called by Identity Manager • 39, 43
 - calling sequence • 43
 - compiling • 149
 - data persistence • 41
 - objects • 41
 - option list • 46
 - purpose • 39
 - sample • 44

- self-registration extension • 50
- use cases • 40
- logical attribute handlers
 - configuring • 45
 - executed before business logic task • 43, 72
 - extended use case example • 57
 - object type • 34
 - purpose • 32
 - sample • 44
 - sharing data with other objects • 41
- logical attributes
 - about • 31
 - adding to task screens • 31, 57
 - alternative validation methods • 39
 - elements of • 34
 - extended use case example • 57
 - fields and • 31
 - name • 55
 - physical attributes and • 33
 - predefined • 35, 103
 - purpose • 32
 - referencing in managed objects • 113
 - regular expression validation • 39
 - summary • 55
- LogicalAttributeAdapter class • 41
- LogicalAttributeContext interface • 41

M

- managed objects
 - about • 109
 - accessing data in • 114
 - accessing through events • 125, 126
 - action type • 144
 - and event listeners • 88
 - attribute names • 113
 - attributes • 12, 113, 114
 - business logic task handlers • 70
 - committing changes • 114
 - integration with third-party applications • 96
 - interfaces • 110, 112
 - modifying in a task session • 121
 - permissions • 119, 140
 - provider access • 114
 - read/write access • 114
 - referencing attributes • 113
 - retrieving attributes • 113, 125
 - retrieving through events • 125, 126
 - setting attributes • 113

- specifying attributes to include • 119
- subject of a task • 121
- superinterfaces • 111
- task session • 65, 121
- task session access • 123
- type • 144
- validation of changes • 145
- ManagedObject interface • 111
- membership
 - constraints • 143
 - policies • 141
 - roles • 112
- messages
 - localization • 137
 - storing exceptions • 138
- ModifiableObject interface • 111, 120
- modifyObject() • 111, 114, 117, 120
- modifyObject(Task[]) • 110, 120
- Multi-Select • 47

N

- N search depth type • 144
- NamedObject interface • 111
- Notification Rule API
 - access to managed objects • 114
 - called by Identity Manager • 103
 - compiling • 149
 - objects • 105
 - sample • 106
 - task example • 23
 - use cases • 104
- notification rules
 - about • 103
 - configuring • 107
 - predefined • 35, 103
 - sample • 106
- NotificationRuleAdapter class • 105
- NotificationRuleContext interface • 105, 124

O

- object type • 144
 - logical attribute handler • 34
- objects
 - events • 124, 125, 128
 - subject of a task • 121
 - types • 144
- ObjectType class • 144
- oldPassword logical attribute • 35
- OperatorType class • 144

- option list
 - about • 46
 - adding to a task screen • 47
 - configuring • 46, 48
 - populating • 46, 49
 - source of list items • 46
- order of execution
 - global and task-specific task handlers • 72
- organization events • 128
- Organization interface • 110
- Organization Selector • 50
- OrganizationEvent interface • 126
- organizations
 - constraints • 143
 - identifier • 98
 - managed object • 110
 - parent • 126
 - specifying, in self registration • 49
- OrgResolverAdapter obsolete • 99
- OrgResolverContext obsolete • 99
- OrgSelector class • 50
- OrgSelectorAdapter class • 50

P

- ParentOrganizationEvent interface • 126
- Participant Resolver API
 - access to managed objects • 114
 - calling sequence • 101
 - compiling • 150
 - objects • 99
 - sample • 101
 - task example • 23
- participant resolvers
 - about • 97
 - configuring • 102
 - custom • 98
 - finding workflow approvers • 97, 101
 - order of precedence • 101
 - organization names • 97
 - sample • 101
 - types • 101
- ParticipantResolverAdapter class • 99
- ParticipantResolverContext interface • 99, 124
- participants
 - finding • 97, 101
 - See also participant resolvers, Participant Resol • 101
- %PASSWORD% • 35
- password policies • 110

- PasswordCondition class • 140
- passwordConfirm logical attribute • 35
- PasswordMessageType class • 144
- PasswordPolicy interface • 110
- passwords
 - invalid • 140, 144
 - password policy managed object • 110
- permissions • 119, 140
- PermissionType class • 144
- persisting data between objects
 - Business Logic Task Handler AP • 67
 - Logical Attribute API • 41
 - Workflow API • 94
- phases
 - asynchronous • 19, 20
 - overview • 15
 - synchronous • 15, 16
- physical attributes
 - about • 31
 - adding to task screens • 31, 57
 - fields and • 31
 - logical attributes and • 33
 - referencing in managed objects • 113
 - where defined • 34
- policy conditions • 141
- populating an option list • 46, 49
- Post requests • 152, 162, 163
- predefined logical attributes • 35
- primary object. See subject of a task • 121
- profile • 109, 121
- profile tabs • 121
- PropertyDict interface • 141
- ProviderAccessor interface • 120
 - managed object access • 114, 117, 118
- providers
 - defined • 117
 - directory-level validation • 145
 - events not generated • 117
 - managed object access • 114
- provisioning
 - eTrust Admin • 109, 110, 142
 - identity policies • 136
 - roles • 142
- ProvisioningRole interface • 110
- ProvisioningRoleEvent interface • 126

Q

- question and answer logical attributes • 35
- Questions logical attribute • 35

R

- read and write permissions • 119
- read/write access • 114, 124
- rejected() • 81, 85
- relationships
 - about • 121
 - adding and removing • 123
 - ResultsContainer class • 142
 - tabs and • 121
- resource bundles
 - base name • 138
 - custom • 138
 - deploying • 150
 - exceptions • 74, 137
 - IMSResources • 140
 - locale • 139, 140
 - name format • 138
 - storing • 138
- ResultsContainer class • 142
- retrieving
 - managed object attributes • 113, 125
 - managed objects through events • 126
- role assignment events • 128
- Role-Based Access Control • 110
- role events • 128
- role grant events • 128
- Role interface • 110
- Role type participant resolver • 97
 - order of precedence • 101
- RoleDisplayData interface • 143
- RoleObjectQuery class • 143
- roles
 - display data • 143
 - managed object • 110
 - membership • 112
 - subject's relationships • 121
 - tabs and • 121
- rule-based assignments
 - identity policies • 136
 - provisioning • 142
- rules
 - constraints • 143
 - policy conditions • 141

S

- samples
 - Business Logic Task Handler API • 69
 - Logical Attribute API • 44
 - Notification Rule API • 106

- Participant Resolver API • 101
- self-registration handler • 52
- Workflow API • 95
- scope of a search • 143
- scripts, workflow • 91
 - sharing data with custom objects • 94
- search depth • 144
- search expressions
 - conjunction type • 144
 - defining • 143
 - operator type • 144
- search operations
 - API objects • 143
 - filters and constraints • 143
- SearchCursor class • 143
- SearchDepthType class • 144
- SearchExpression class • 143
- secondary events
 - generating • 114, 124
 - generating from workflow script • 94
- security checks
 - available in a task session • 122
 - unavailable with providers • 117
- Self Registration
 - specifying the organization • 49
- Self Subscribing Handler • 35
- self-registration events • 128
- self-registration handlers
 - and logical attribute handlers • 50
 - coding • 50
 - purpose • 50
 - sample • 52
- %SELF_SUBSCRIBING% • 35
- session. See task session • 64
- setAttribute... methods and attribute validation
 - 123
- setting managed object attributes • 113
- Single-Select • 47
- SiteMinder Directory • 109
- skins, localizing • 140
- SOAP document • 152, 154, 159, 161, 162, 166, 171, 182
- StringRef • 145
- subject of a task
 - about • 121
 - modifying • 123
 - modifying in a task session • 123
 - relationships • 121, 142
 - tabs and • 121

- submitting a task • 137
- support
 - support, contacting • iii
- synchronous phase • 15, 16, 25

T

- task events • 128
- Task Execution Web Service
 - configuring • 156, 157
 - development overview • 159
 - exceptions • 162, 182
 - generated files • 173
 - operation types • 154, 156
 - overview • 152
- Task.FieldId class • 144
- Task interface • 110
- task-level validation • 145
- task screen
 - attribute validation • 145
 - attributes and • 31
 - business logic task handler and • 121
 - data flow • 32
 - exception messages • 74, 137
 - field validation • 145
 - localized labels • 139
 - logical attributes • 16, 55
 - option list • 47
 - tabs • 121
- task session
 - about • 64
 - attribute validation • 145
 - contents • 121
 - event generation • 122
 - field validation • 145
 - managed objects • 65, 121
 - modifying objects in • 121
 - overview • 15
- TaskObjectQuery class • 143
- tasks
 - associating with a role • 110, 120
 - events in • 79, 124
 - exceptions • 162, 182
 - execution phases • 15
 - execution summary • 23
 - in asynchronous operations • 20
 - in synchronous operations • 16
 - localized names • 139
 - managed object • 110
 - operation flow • 156

- operation types • 154, 156
- overview • 15
- POST request • 152
- remote • 152, 153
- session • 64
- SOAP • 152, 154, 159, 161, 162, 166, 171, 182
- subject of • 121, 142
- submitting • 137
- workflow controlled • 20
- task-specific business logic task handlers
 - order of execution; • 72
 - about • 66
- TaskValidator interface • 145
- TaskValidator.StringRef class • 145
- technical support
 - technical support, contacting • iii
- templates for email • 103
- third-party applications
 - external tasks • 126
 - integration with • 96
- toLogical() • 43
- toPhysical() • 16, 43
- trigger rules. See policy conditions • 141
- TSContext interface • 144

U

- user certification events • 128
- user directory • 31
- user events • 128
- User interface • 110
- USER notification rule • 103
- USER object type • 35
- USER_MANAGER notification rule • 103
- user-defined data
 - option lists • 46
- UserEvent interface • 126
- users
 - admin overview • 15
 - managed object • 110
 - self-registered • 49

V

- validate() • 16, 43
- validation
 - directory-level • 145
 - task-level • 145
- validation rules
 - on attributes • 123

- types of • 145
- VerifyQuestion logical attribute • 35
- ViewUserQuery • 154
- ViewUserQueryResult • 154

W

- web service • 152, 159, 161, 171
- web.xml file • 164
- well-known attributes
 - mapped to logical attributes • 34
 - mapped to physical attributes • 34
 - referencing in managed objects • 113
- workflow
 - approval task • 12, 99
 - event generation • 94, 122
 - scripts • 91
- workflow activities
 - finding participants • 101
 - participants • 97
- Workflow API
 - access to managed objects • 114
 - and compilation • 149
 - called by workflow scripts • 92
 - data persistence • 94
 - generating secondary events • 94
 - integration with third-party applications • 96
 - objects • 94
 - sample • 95
 - task example • 23
 - use cases • 92
- WorkflowContext interface • 94, 124
- WSDL • 152, 159