CA Identity Manager™

Programming Guide for Java

12.6.4

This Documentation, which includes embedded help systems and electronically distributed materials, (hereinafter referred to as the “Documentation”) is for your informational purposes only and is subject to change or withdrawal by CA at any time. This Documentation is proprietary information of CA and may not be copied, transferred, reproduced, disclosed, modified or duplicated, in whole or in part, without the prior written consent of CA.

If you are a licensed user of the software product(s) addressed in the Documentation, you may print or otherwise make available a reasonable number of copies of the Documentation for internal use by you and your employees in connection with that software, provided that all CA copyright notices and legends are affixed to each reproduced copy.

The right to print or otherwise make available copies of the Documentation is limited to the period during which the applicable license for such software remains in full force and effect. Should the license terminate for any reason, it is your responsibility to certify in writing to CA that all copies and partial copies of the Documentation have been returned to CA or destroyed.

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 YOU OR ANY THIRD PARTY FOR ANY LOSS OR DAMAGE, DIRECT OR INDIRECT, FROM THE USE OF THIS DOCUMENTATION, INCLUDING WITHOUT LIMITATION, LOST PROFITS, LOST INVESTMENT, BUSINESS INTERRUPTION, GOODWILL, OR LOST DATA, EVEN IF CA IS EXPRESSLY ADVISED IN ADVANCE OF THE POSSIBILITY OF SUCH LOSS OR DAMAGE.

The use of any software product referenced in the Documentation is governed by the applicable license agreement and such license agreement is not modified in any way by the terms of this notice.

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.

Copyright © 2014 CA. All rights reserved. All trademarks, trade names, service marks, and logos referenced herein belong to their respective companies.

CA Technologies Product References

This document references the following CA products:

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

Contact CA Technologies

Contact CA Support

For your convenience, CA Technologies provides one site where you can access the information that you need for your Home Office, Small Business, and Enterprise CA Technologies products. At http://ca.com/support, you can access the following resources:

- Online and telephone contact information for technical assistance and customer services
- Information about user communities and forums
- Product and documentation downloads
- CA Support policies and guidelines
- Other helpful resources appropriate for your product

Providing Feedback About Product Documentation

If you have comments or questions about CA Technologies product documentation, you can send a message to techpubs@ca.com.

To provide feedback about CA Technologies product documentation, complete our short customer survey which is available on the CA Support website at http://ca.com/docs.

Contents

Chapter 1: CA Identity Manager Architecture 13

Architecture Diagram 13
Servers 14
CA Identity Manager Server 15
Provisioning Server 15
User Stores 16
Databases 16
Managed Endpoints 17
Connectors 17
Connector Servers 17
CA IAM Connector Server 17
C++ Connector Server 18
Connectors and Agents 18
Additional Components 19
Provisioning Manager 19
WorkPoint Workflow 19
Report Server 19
Development Environment Prerequisites 20
System Recommendations 20
Knowledge Recommendations 20

Chapter 2: CA Identity Manager APIs 21

API Installation 21
API Overview 21
CA Identity Manager APIs 22
Types of Custom Operations 23
Task Phases 24
Synchronous Phase 24
Synchronous Phase Operations 25
Asynchronous Phase 28
Asynchronous Phase Operations 29
Task Execution Summary 32
API Functional Model 34
Synchronous Phase Model 34
Asynchronous Phase Model 35
Model Components 36

Chapter 3: Logical Attribute API 39

Logical Attributes and Physical Attributes 39
Logical Representation of Physical Attributes 40
Logical Attribute Elements 41
Attribute Data Flow 42
Predefined Logical Attributes 43
Password 43
Password Hint 44
Password Reset 44
Enable User 45
Group Subscription 45
Logical Attribute API Overview 46
Logical Attribute API Summary 46
Use Case Examples 47
API Components 48
Logical Attribute Handler API Model 49
Calling Sequence 50
Custom Messages 51
Sample Logical Attribute Handlers 51
Logical Attribute Implementation 52
Logical Attribute Handler Configuration 52
Logical Attribute Handler at Run Time 53
Example: Assigning a Logical Attribute to a Field 53
Use Case Example: Formatting a Date Attribute 54
Screen-Defined Logical Attributes 57
Display the Configure Standard Profile Screen 58
Initializing Screen-Defined Logical Attributes 58
Updating Screen-Defined Logical Attributes 59
Logical Attribute Handler: Option List 59
Logical Attribute Handler: Self-Registration 62
How a Self-Registration Handler Works 63
Writing the Self-Registration Handler 64
Self-Registration and the API Functional Model 65
Self-Registration Handler Configuration 65
Sample Self-Registration Handler 66
Logical Attribute Handler: Forgotten Password 66
Writing a Custom Forgotten Password Handler 67
Forgotten Password Handler Configuration 68

Chapter 4: Business Logic Task Handler API 69

Business Logic Task Handler API Overview 69

Business Logic Task Handler API Summary 69
Task Sessions 70
Use Case Examples 72
Scope of Business Logic Task Handlers 73
Configuring Business Logic Task Handlers 73
How to Configure a Global BLTH 74
How to Configure a Task-Specific BLTH 74
API Components 75
Business Logic Task Handler API Model 76
Sample Business Logic Task Handlers 76
Calling Sequence 77
Example: Calling a Business Logic Task Handler 78
Custom Messages 78
Informational Messages 78
Exception Messages 79
Custom Message Example 79
Access to Managed Objects 80
Managed Object Process Diagram 81
JavaScript Business Logic Task Handlers 81
JavaScript Method Reference 82
IMSAPI Class References 83
Default Installed Handlers 83

Chapter 5: Event Listener API 85

Tasks and Events 85
Event Listener API Overview 86
Event Listener Operations 86
Use Case Examples 88
API Components 89
Event Listener API Model 90
Event Listener Execution 90
Event Listener States 91
Event Listener State Diagram 92
Sample Event Listener 93
Order of Execution 93
Access to Managed Objects 94
Event Generation 95
Event Listener Configuration 95

Chapter 6: Validation Rules 97

Validation Rules Introduction 97

About Validation Rules 97
Types of Validation Rules 98
Validation Rule Sets 99
Basics of Validation Rule Definition 100
Using Default Validation Rules 101
Default Data Validations 101
Predefined Validation Rules 103
How to Implement Custom Validation Rules 104
Regular Expression Implementation 104
JavaScript Implementation 105
Java Implementation 107
Exceptions 110
How to Configure Validation Rules 112
How to Configure Task-Level Validation 112
How to Configure Directory-Level Validation 113
How to Initiate Validation 118
Sample Implementations 119

Chapter 7: Workflow API 121

Workflow API Overview 121
Workflow API Summary 121
Use Case Examples 122
API Components 123
Integration with Third-Party Applications 125
Saving Job Objects 125
Participant Resolver API 126
Participant Resolver Overview 127
API Components 129
Calling Sequence 130
Participant Resolver Configuration 131

Chapter 8: Notification Rule API 133

API Overview 133
Notification Rule API Summary 133
Use Case Examples 134
API Components 134
Notification Rule API Model 135
Sample Notification Rule 136
Email Templates 136
Notification Rule Configuration 136

Chapter 9: Managed Objects 139

Managed Object Overview 139
Managed Object Interfaces 140
Super Interfaces 142
Extended Interfaces 142
Managed Object Attributes 143
Retrieval Example 143
Access to Managed Object Data 144
ProviderAccessor 145
BLTHContext 145
EventContext 146
EventROCContext 146
Access to Objects in the Data Store 146
Providers 147
Retrieving Providers 148
Attribute and Permission Requests 148
Access to Attribute Data 149
Calling Sequence 149
Access to Objects in a Task 150
Parallels with Task Screen Operations 150
Retrieval of Managed Objects 151
Calling Sequence 152
Attribute Validation Type 152
Access to Event Objects 153
Events and Event Objects 153
Event Object Tables 157

Chapter 10: Support Objects 159

Identity Policies 159
IMEvent 160
IMEventName 160
IMSException 161
Localizing Exception Messages 161
Localizer Class 163
Use the Localizer Class 163
PasswordCondition 163
Permissions Objects 164
Policies and Policy Conditions 164
PropertyDict 165
Provisioning Roles 165
ResultsContainer 166

RoleDisplayData 166
Search Objects 167
Search Filters and Constraints 167
Task.FieldId 168
TSContext 168
Type Objects 168
Validation Objects 170
Attribute Validation 170

Chapter 11: Compiling and Deploying 173

Compiling Source Files 173
Deploying Run Time Files 174

Chapter 12: Task Execution Web Service 175

Web Services Overview 175
Web Service Protocols 175
About TEWS 176
Remote Task Requests 177
How a Remote Request Is Processed 178
Task Operation Types 179
How Task Operations Work 180
A Task Operations Example 180
Web Service Configuration 181
Configure Web Service Properties 181
Web Services Properties Screen 182
Enable Web Services for Individual Tasks 183
Client Application Development 184
WSDL Generation 186
SOAP Messages 187
SOAP Limitations 187
Exception Messages 187
Authentication Using SiteMinder 188
Authentication Using WSS Username Token 189
Authorization 189
Administrator Specification 190
Administrator Parameters 191
The Channel Connection 192

Chapter 13: TEWS Sample Client Code 193

Web Service Development Workstation 193

Third-Party Software 193
Web Service Samples 194
Java Samples for Axis 195
Java Sample Classes 195
Build Script for Axis 196
Running the Java Classes 196

Appendix A: TEWS Error Codes 197

Task Session Error Codes 198

Appendix B: Custom Authentication Schemes 199

How To Customize CA Identity Manager Authentication 199
Modify the Login Credential Form 199
Implement the AuthenticationModule Interface 200
Configure the Java Class and Login Page 204

Index 207

Chapter 1: CA Identity Manager Architecture

This section contains the following topics:

Architecture Diagram (see page 13)
Servers (see page 14)
User Stores (see page 16)
Databases (see page 16)
Managed Endpoints (see page 17)
Connectors (see page 17)
Connector Servers (see page 17)
Additional Components (see page 19)
Development Environment Prerequisites (see page 20)

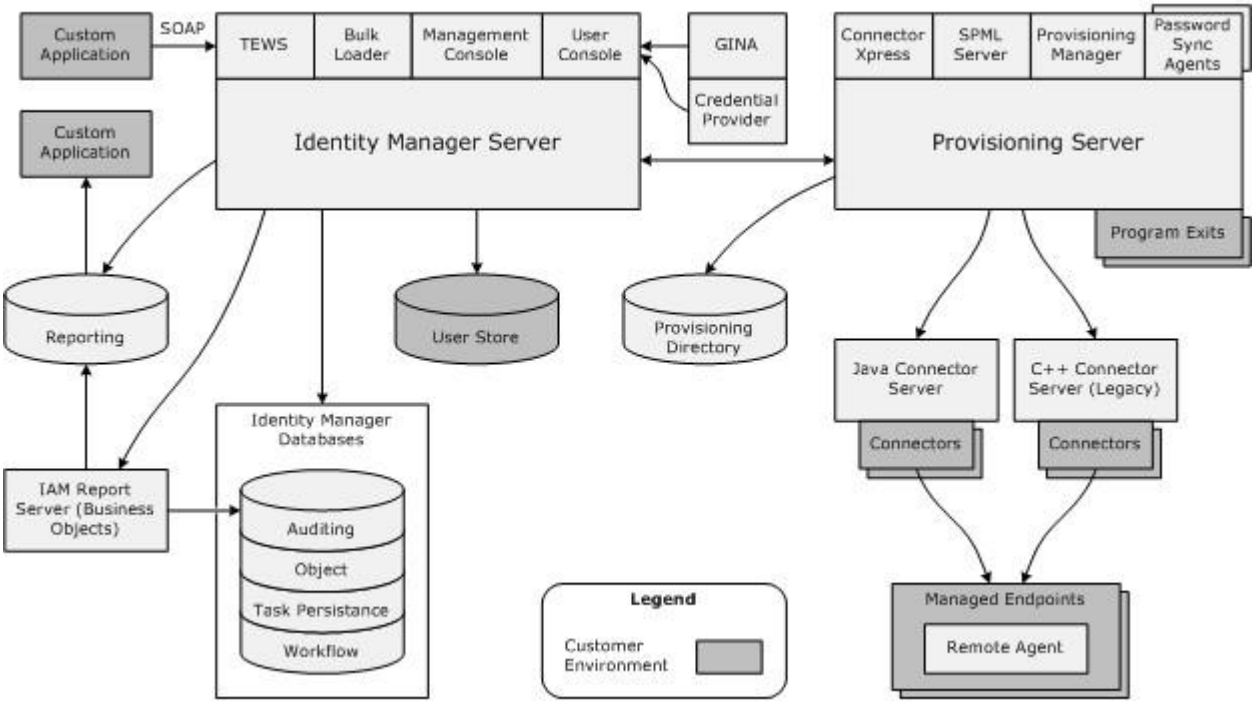
Architecture Diagram

A typical CA Identity Manager installation includes the following components:

- Servers
- User Stores
- Databases
- Connectors

Depending on how it is configured, your installation can have one or more of these modules.

A CA Identity Manager implementation can include some or all of the following components:



Servers

A typical CA Identity Manager installation includes the following servers:

CA Identity Manager Server

Executes tasks within CA Identity Manager. The CA Identity Managerr application includes the Management Console and the User Console.

Provisioning Server

Manages accounts on endpoint systems. This is required if the CA Identity Manager installation supports account provisioning.

Note: You must have the Provisioning Directory installed remotely (or locally for a demonstration environment only) on a CA Directory Server before installing the Provisioning Server.

CA Identity Manager Server

The CA Identity Manager Server provides an integrated method of managing users and their access to applications, including:

- Assignment of privileges through provisioning roles, including the following:
  - Roles that enable administrators to create and maintain user accounts
  - Roles that provide application rights to users
  - Roles that provision additional accounts to existing users (requires integration with the Provisioning Server)
- Delegation of the management of users and application access
- Self-service options so users can manage their own accounts
- Integration of business applications with CA Identity Manager
- Options to customize and extend CA Identity Manager

Provisioning Server

The Provisioning Server consists of the following components:

- OpenLDAP (SLAPD) server configuration
- CA Directory configuration
- Provisioning Server back end to SLAPD
- Superagent back end to SLAPD
- Superagent Server Plug-in (SASP)
- C++ SDK Sample Connector
- Program Exit samples

The Provisioning Server handles the provisioning of users, and then delegates to connectors (using the C++ Connector Server or CA IAM CS) to manage endpoint accounts, and groups.

The Provisioning Server acts as the manager and command center for all communication.

User Stores

CA Identity Manager includes two user stores:

CA Identity Manager User Store

Typically, this is an existing store that contains the user identities that a company needs to manage. The user store can be an LDAP directory or a relational database.

Provisioning Directory

This is an instance of CA Directory and includes global users, which associate users in the Provisioning Directory with accounts on managed endpoints such as Microsoft Exchange, Active Directory, and Oracle.

Databases

CA Identity Manager uses data sources to connect to databases that store information required to support CA Identity Manager functionality. These databases can reside in a single physical instance of a database, or in separate instances. By default, CA Identity Manager configures a connection to a single database, called the CA Identity Manager Database, which contains the tables for each database type.

Object Store Database (required)

Contains CA Identity Manager configuration information.

Task Persistence Database (required)

Maintains information about CA Identity Manager activities and their associated events over time. This system tracks all activities, even if you stop and restart the CA Identity Manager Server.

Workflow Database

Stores workflow process definitions, jobs, scripts, and other data required by the workflow engine.

Audit Database

Provides a historical record of operations that occur in a CA Identity Manager environment.

Archive Database

Stores archived task persistence data.

Snapshot Database

Stores snapshot data, which reflects the current state of objects in CA Identity Manager at the time the snapshot is taken. You can generate reports from this information to view the relationship between objects, such as users and roles.

Managed Endpoints

An endpoint type is a specific type of target system. There is typically one connector server plug-in for each endpoint type, but there can be multiple endpoints for each endpoint type.

Generic Java plug-ins (JNDI, JDBC) are each able to manage multiple endpoint types.

Examples of endpoints are UNIX workstations, Windows PCs, or applications such as Microsoft Exchange.

Connectors

A connector is the software that enables communication between a Provisioning Server and an endpoint system. The code that makes up a connector is contained in a connector server plug-in.

A dynamic connector can be generated by Connector Xpress, and a custom static connector can be developed in Java or C++.

Connector Servers

A connector server is a Provisioning Server component that manages connectors. It can be installed on the Provisioning Server system or on a remote system.

A connector server works with multiple endpoints. For example, if you have many UNIX workstation endpoints, you might have one connector server which handles all connectors that manage UNIX accounts. Another connector server might handle all connectors that manage Windows accounts.

There are two types of connector servers:

- CA IAM Connector Server (CA IAM CS) manages connectors written in Java
- C++ Connector Server manages connectors written in C++

CA IAM Connector Server

CA IAM Connector Server (CA IAM CS) handles hosting, routing to, and management of Java connectors. It can be installed on the Provisioning Server or on a remote system. CA IAM CS is data-driven rather than code-driven, which allows more functionality to be addressed by the container instead of by connectors themselves.

C++ Connector Server

The C++ Connector Server handles hosting, routing to, and management of C++ connectors.

Note: This is a legacy component. New connectors should be written to execute in CA IAM CS.

Connectors and Agents

CA Identity Manager connectors run as part of the Provisioning Server and communicate with the systems managed in your environment. A connector acts as a gateway to a native endpoint type system technology. For example, machines running Active Directory Services (ADS) can be managed only if the ADS connector is installed on a connector server with which the Provisioning Server can communicate. Connectors manage the objects that reside on the systems. Managed objects include accounts, groups, and optionally, endpoint type specific objects.

Some connectors require agents on the systems they manage to complete the communication cycle. The two categories of agents are as follows:

- Remote agents are installed on the managed endpoint systems.
- Environment agents are installed on systems, such as CA ACF2, CA Top Secret, and RACF.

The following C++ connectors work on UNIX and Windows:

- UNIX (ETC, NIS)
- Access Control (ACC)

  Note: The UNIX ACC connector can manage only UNIX ACC endpoints. The Windows ACC connector is required to manage Windows ACC endpoints but can also manage UNIX ACC endpoints.

- CA ACF2
- RACF
- CA Top Secret

Other C++ connectors can be accessed from a Provisioning Server installed on Solaris, by relying on the Connector Server Framework (CSF). The CSF allows a Provisioning Server on Solaris to communicate with connectors running on Windows.

Note: The CSF must run on Windows to use these connectors.

Additional Components

CA Identity Manager includes additional components that support particular functionality. Some of these components are installed with CA Identity Manager, and some must be installed separately.

Provisioning Manager

The Provisioning Manager, an alternative to the User Console, manages the Provisioning Server through a graphical interface. The Provisioning Manager can be used for administrative tasks such as acquiring endpoints, installing endpoint types, and managing Provisioning Server options.

The Provisioning Manager is installed as part of the CA Identity Manager Administrative Tools.

Note: This application runs on Windows systems only.

WorkPoint Workflow

The WorkPoint workflow engine and WorkPoint Designer are installed automatically when you install CA Identity Manager.

These components enable you to place a CA Identity Manager task under workflow control, and to modify existing workflow process definitions or create new process definitions.

Note: For more information about workflow, see the *Administration Guide*.

Report Server

CA Identity Manager provides reports that you can use to monitor the status of a CA Identity Manager Environment. To use the reports provided with CA Identity Manager, install a supported version of CA Business Intelligence, which can be downloaded from the CA support site.

The Report Server is powered by Business Objects Enterprise XI. If you have an existing Business Objects server, you can use it to generate CA Identity Manager reports.

Note: For installation instructions, see the *Installation Guide*.

Development Environment Prerequisites

A CA Identity Manager development environment must conform to the following requirements:

- Provisioning Server is installed. By default, the Provisioning Server is installed in the following directory:

  \Program Files\CA\Identity Manager\Provisioning Server

- The Provisioning SDK is installed. By default, the SDK is installed in the following directory:

  \Program Files\CA\Identity Manager\Provisioning SDK

  Note: The Provisioning SDK can be installed on a system other than the Provisioning Server itself.

- Set the JAVASDK environment variable to point to the installed JAVA SDK location. This setting is required to build the JIAM Jar files for connectors.
- Microsoft Visual C++ 2008, Version .NET, with MFC Shared Libraries for Unicode.
- Unicode libraries must be installed (MFC71u.dll)

System Recommendations

A typical CA Identity Manager development system on Windows, as recommended by CA, has the following components on three separate disk drives:

- The Microsoft Windows operating system (on drive C:)
- The Provisioning Server (on drive D:)
- Microsoft Visual Studio 2008
- The Provisioning SDK and custom connectors (on drive E:)

This is to optimize the combination of running an X.500 server CA Directory and Provisioning Server on the same host.

Knowledge Recommendations

You should have a working knowledge of the following:

- The CA Identity Manager architecture and user interface
- Developing and building C++ applications
- Developing and building Java applications
- Developing and building applications that use Web Services

Chapter 2: CA Identity Manager APIs

This section contains the following topics:

API Installation (see page 21)
API Overview (see page 21)
CA Identity Manager APIs (see page 22)
Types of Custom Operations (see page 23)
Task Phases (see page 24)
Task Execution Summary (see page 32)
API Functional Model (see page 34)

API Installation

Install the CA *I*dentity Manager APIs by installing the CA Identity Manager Administrative Tools, described in the *Installation Guide*.

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

- **Windows:** C:\Program Files\CA\Identity Manager\IAM Suite\Identity Manager\tools
- **UNIX:** /opt/CA/IdentityManager/IAM_Suite/Identity_Manager/tools

Note: The notation *admin_tools* is used as a convenience throughout this guide to refer to the Administrative Tools installation directory.

API Overview

You can use the CA Identity Manager APIs to customize the following features:

- The schema, which describes a user store's structure to CA 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 Task Execution Web Service (TEWS). TEWS processes the remote task request. Remote task requests comply with WSDL standards.

CA Identity Manager APIs

You can use the following APIs to customize CA 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 displayed, convert an attribute value from its stored physical form to its displayed form
- To populate the value of an attribute automatically, possibly through an external system
- To 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 CA Identity Manager event being executed. Using this information, a workflow script can decide 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 CA 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 to 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 *Administration Guide*. For technical reference information about all CA Identity Manager APIs, see the *Programming Guide for Java*.

Types of Custom Operations

The following table lists the kinds of operations you can perform with CA 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 a CA 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

An admin *task* is an administrative operation, made up of one or more events, performed by CA Identity Manager administrators to manage or modify CA Identity Manager objects such as users, roles, groups, organizations, endpoints, and other objects.

Admin task execution has the following 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.

Note: For more information about admin tasks, see the *Administration Guide*.

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 for processing.

Synchronous Phase Processing

When a user logs in, CA 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 CA 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 139)

Synchronous Phase Operations

In the synchronous phase, the following operations are performed:

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 subject is selected but not yet displayed.

   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.

   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

   - 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.

3. CA Identity Manager displays and populates the task screen

   After the user has selected the task to perform and the task's subject, CA 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. CA Identity Manager invokes validation rules during data entry.

   CA Identity Manager can 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 the following 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, CA Identity Manager performs the following operations in order:

   a. Invokes validation rules.

      Validation rules can be implemented as regular expressions, JavaScript, or Java classes.

   b. 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

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

   API: Logical Attribute API.

d. 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, CA 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. CA 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. CA Identity Manager instantiates events for the task.

   After all validation is performed successfully, CA 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.

After events are instantiated for the task, the following processing can occur to allow a finer degree of control over the execution of a task:

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

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.

API: Business Logic Task Handler API

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

8. 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 *Configuration Guide*.

Asynchronous Phase

The asynchronous phase performs the task by executing the events associated with the task. Optionally, the asynchronous phase performs workflow approvals and sends email notifications. It begins when data validation and manipulation performed in the synchronous phase are complete. It ends when task processing is complete.

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. CA 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. CA Identity Manager posts the events.

   At the beginning of the asynchronous phase of task execution, CA 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 CA 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 CA 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 CA 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 *Administration 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, CA Identity Manager executes the associated event.

   After a workflow process is approved or rejected, but before CA 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. CA 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 CA 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 do the following:
     - 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. CA 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 CA 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

The following process 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 they are authorized to perform.
2. CA Identity Manager creates a task session.

   API: CA Identity Manager calls BLTHAdapter.handleStart() in applicable business logic task handlers (those handlers that are defined in the environment or are associated with the task).
3. The administrator selects a particular User object to modify.

   API: CA Identity Manager calls BLTHAdapter.handleSetSubject() in applicable business logic task handlers.
4. CA Identity Manager retrieves the User object.

   API: If any physical attributes in the User object must be converted to logical attributes, logical attribute handlers perform the conversions.
5. CA 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: CA 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: CA Identity Manager calls BLTHAdapter.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. CA Identity Manager instantiates the events ModifyUserEvent and AddToGroupEvent. In this example, both of these events are mapped to workflow processes.

   API: CA Identity Manager calls BLTHAdapter.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. CA Identity Manager posts the events instantiated in the synchronous phase. Events can no longer be recalled.

9. CA Identity Manager begins processing the event ModifyUserEvent.

   API: CA 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 perform.

11. CA 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, CA Identity Manager calls the approved() or rejected() method in each event listener for the event ModifyUserEvent.

13. CA 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, CA Identity Manager executes the event.

    API: After executing the event, CA 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, CA Identity Manager completes the Modify User task.
17. CA 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 CA 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.

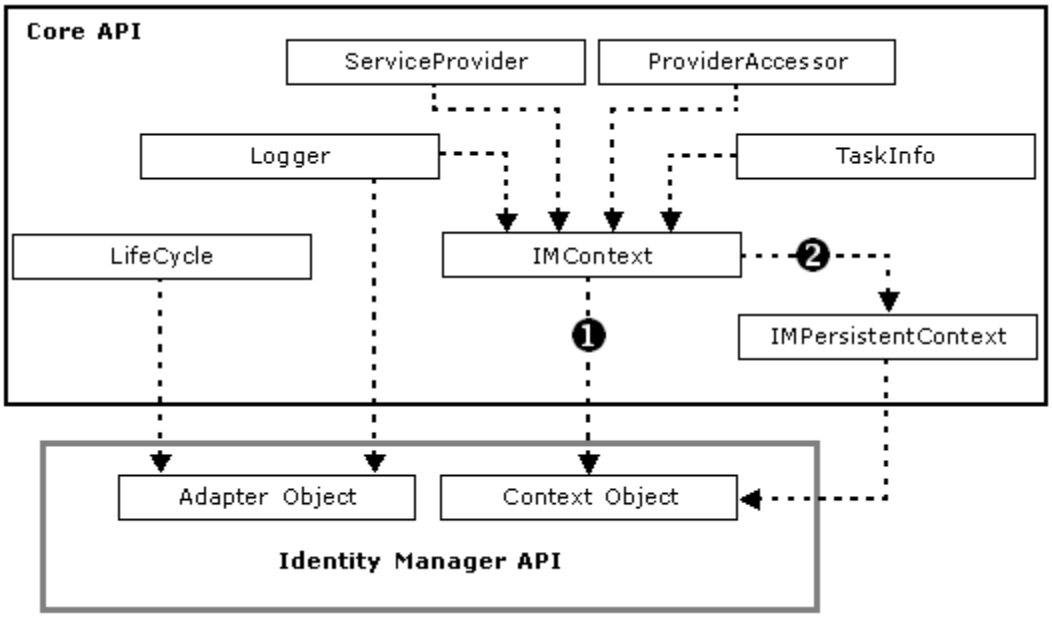
The following CA Identity Manager APIs can be used during this phase:

- **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).

Each API inherits methods from the core IMContext interface in different ways.

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

Equation 1: Diagram shows the core classes and interfaces in the Synchronous Phase 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 include the following:

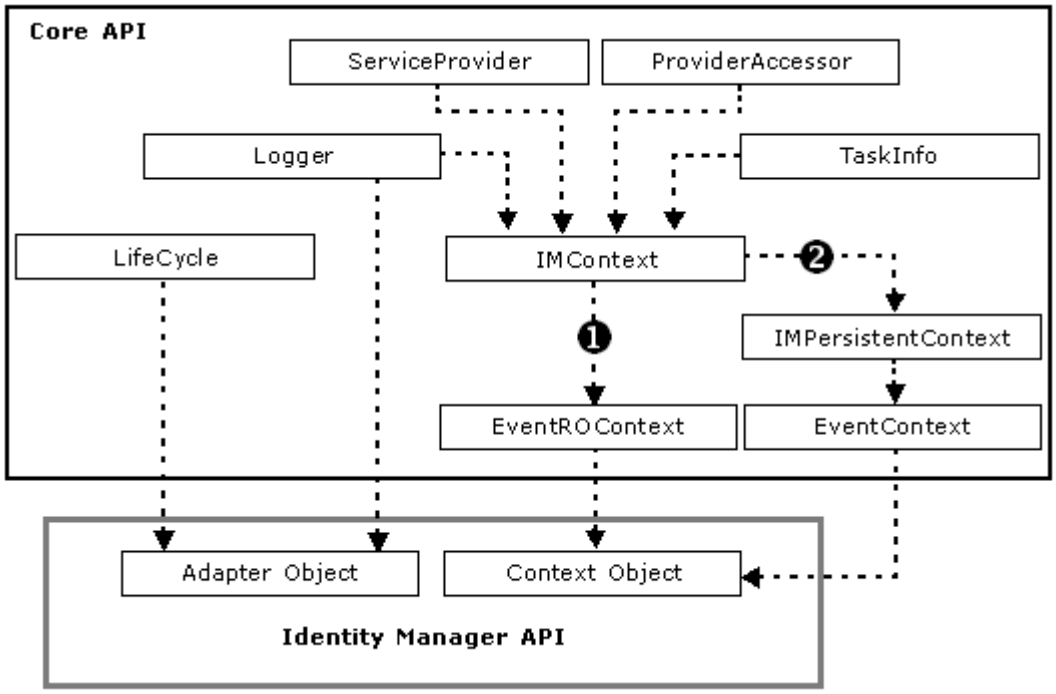
- **EventContext**—Provides read/write access to events and lets you generate secondary events.
- **EventROContext**—Provides read/write access to events.

The following CA Identity Manager APIs can be used during this phase:

- **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 APIs inherit methods from the core IMContext interface in different ways.

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 Management Console. The properties can be applicable to a single custom object, to the entire CA Identity Manager environment, or both.

  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 CA 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 CA Identity Manager APIs implement the LifeCycle interface.

- **ServiceProvider interface**—Provides CA Identity Manager APIs with access to services such as password and encryption operations.

- **ProviderAccessor interface**—Makes the CA 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 Management Console for a particular custom object or for the CA 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 CA Identity Manager APIs typically have the following types of components:

- **Adapter object**—Each CA 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 CA 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 CA 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 CA Identity Manager services, including managed object retrieval from the data store.

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

Chapter 3: Logical Attribute API

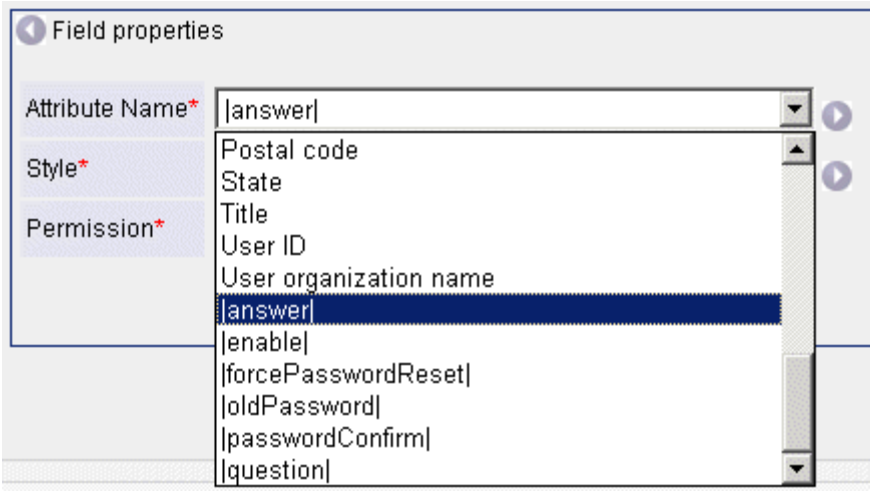
This section contains the following topics:

Logical Attributes and Physical Attributes (see page 39)
Predefined Logical Attributes (see page 43)
Logical Attribute API Overview (see page 46)
Logical Attribute Implementation (see page 52)
Screen-Defined Logical Attributes (see page 57)
Logical Attribute Handler: Self-Registration (see page 62)
Logical Attribute Handler: Forgotten Password (see page 66)

Logical Attributes and Physical Attributes

In the User Console, 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:



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 can be written to the data store if the logical attribute is mapped to a physical attribute, or the result can 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 53)

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 CA 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 drop-down list, and therefore can be assigned to a task screen field.

More Information:

Logical Attributes and Physical Attributes (see page 39)
Logical Attribute Handlers for Option Lists (see page 60)

Attribute Data Flow

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

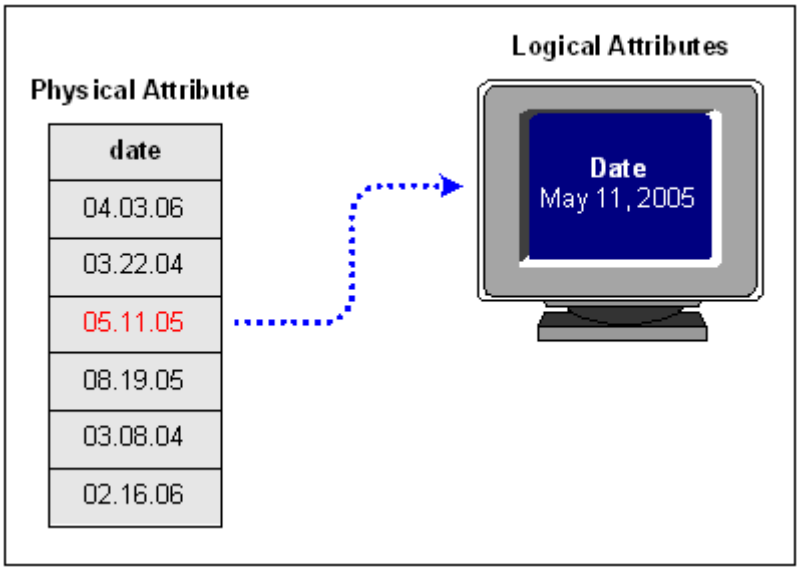
In a simple data flow, a task screen field is associated with a physical attribute. Data in this field flows to and from the corresponding physical attribute in the data store. In this case, no logical attribute handlers are involved in the data flow between the task screen and the data store.

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.

Channeling 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 own business logic and rules with the presentation that users see.

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



Predefined Logical Attributes

CA 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 as follows:

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 be 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 Provisioning: 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 their password during their 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.

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.

Is available for GROUP objects.

If the administrator sets this attribute to TRUE, users can add themselves to the group.

The handler does the following:

- Converts between physical and logical attribute values.
- Validates the self-subscription value before the value is written to the data store.

More Information:

Logical Attribute Handler: Forgotten Password (see page 66)

Logical Attribute API Overview

The Logical Attribute API (javadoc-im/com/netegrity/imapi/LogicalAttributeAdapter.html) 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.

  Note: To perform broader validation operations against information in the entire task, use the Business Logic Task Handler API (javadoc-im/com/netegrity/imapi/BLTHAdapter.html).
- 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. CA Identity Manager then writes the physical attribute values to the data store.

More Information:

Business Logic Task Handler API (see page 69)

Logical Attribute API Summary

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.

Operations

The Logical Attribute API performs the following:

- Display and population of task screens
- Invocation of validation rules during data entry
- User submissions from the task screen.

Called by

CA Identity Manager

When called

Under any of the following circumstances:

- After the user selects a task to perform, and before the task screen appears.
- When 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: You can also perform attribute validation using JavaScript and regular expressions. For more information, see the *User Console Design Guide*.

Use Case Examples

The following are 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 telephone number is about to be displayed, CA Identity Manager calls a custom logical attribute handler that formats the physical attribute telephone number appropriately—for example, as (978) 555-1212—and assigns the formatted value to *formatted_phone*. CA 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, and 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, CA Identity Manager calls a custom logical attribute handler to query the data source to be sure 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 they have purchased—Gold, Silver, or Bronze. These options are presented in a drop-down list associated with the logical attribute *serviceType*. When the customer chooses a service level and clicks Submit, CA 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 54)

API Components

The Logical Attribute API contains the following components:

LogicalAttributeAdapter
(**javadoc-im/com/netegrity/imapi/LogicalAttributeAdapter.html**)

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
(javadoc-im/com/netegrity/imapi/LogicalAttributeContext.html)

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 CA Identity Manager services, including managed object retrieval from the data store.

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

Extends MPersistentContex.

IMPersistentContext (javadoc-im/com/netegrity/imapi/IMPersistentContext.html)

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.

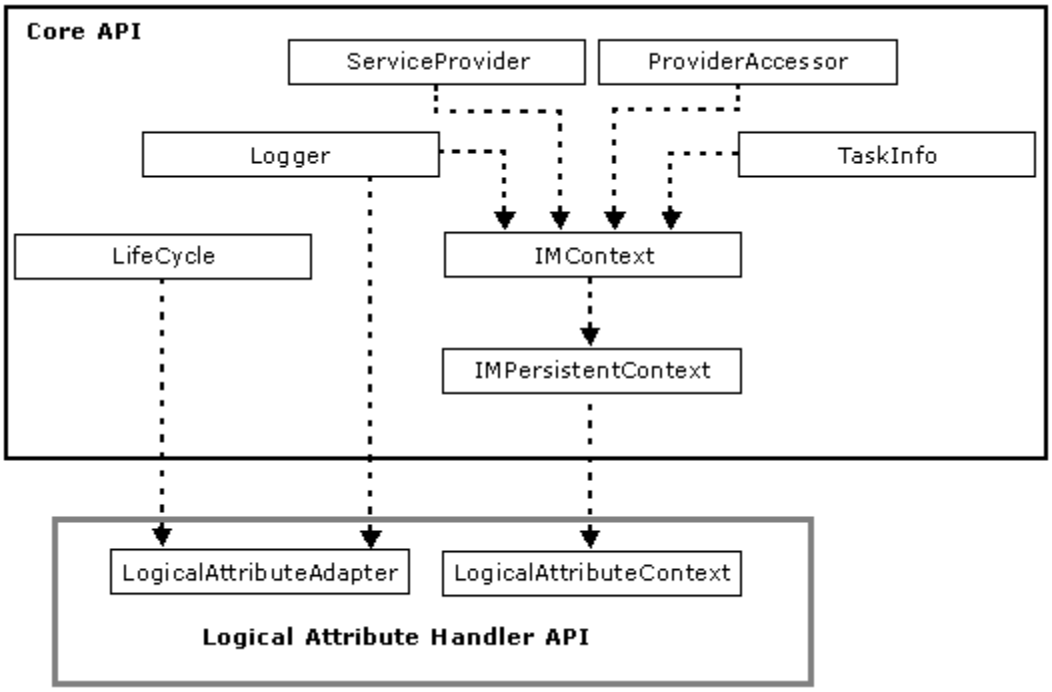
This interface is part of the core API functional model.

Extends IMContext.

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

Logical Attribute Handler API Model

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



Calling Sequence

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

CA Identity Manager calls LogicalAttributeAdapter methods in the following stages of task screen processing:

1. initializeOptionList(). This is the first method CA 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, CA 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.

CA Identity Manager calls logical attribute handlers in the order in which they are listed in the Management Console. In each successive handler, CA 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: CA 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, CA 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 message gives the user the opportunity to correct the errors and resubmit the task.

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

CA 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 CA Identity Manager selects the appropriate resource bundle from which to retrieve exception messages.

Note: For more information, see the IMSException class in the Javadoc Reference.

More Information:

IMSException (see page 161)

Sample Logical Attribute Handlers

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

admin_tools\samples\LogicalAttributes

Note: For information about using the samples, see the accompanying readme.txt files.

Logical Attribute Implementation

The following process lists the high-level actions that must be performed when you create a logical attribute. 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 Management Console for the CA Identity Manager environment.
3. Compile and deploy the handler.
4. Add the logical attribute to a task screen being constructed in the User Console.

More Information:

Logical Attribute Handler at Run Time (see page 53)
Compiling and Deploying (see page 173)
Logical Attribute API Summary (see page 46)
Logical Attribute Handlers for Option Lists (see page 60)

Logical Attribute Handler Configuration

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

To define a logical attribute handler

1. In the Management Console, click 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 CA Identity Manager and any custom handlers defined at your site. CA 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 CA 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 Help on the Logical Attribute Handlers screen.

Note: You can also create, modify, view, and delete a logical attribute handler from the User Console, under the System tab. For more information, see the Online Help.

More Information:

Compiling and Deploying (see page 173)

Logical Attribute Handler at Run Time

After you create and deploy the logical attribute handler, 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. CA 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 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 CA 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: The logical attribute must first be defined in the Management Console. It is not a predefined logical attribute shipped with CA Identity Manager.

To configure a custom logical attribute in a Profile screen

1. In the User Console, select Roles and Tasks, Admin Tasks, Modify Admin Task.
2. Click Search to display a list of all admin tasks.
3. Select the Self Registration option button, then click Select.
4. Click the Tabs tab in the Modify Admin Task Self Registration screen.
5. Click the edit button next to the Profile tab in the list of tabs.
6. Click the browse button to the right of the Screen field in Configure Profile.

   A list of profile screens appears.
7. Be sure that the Self Registration Profile option 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 selecting the check box to the left of Password, then clicking the appropriate Add button.
9. Click the edit button to the right of the newly added field.

   A list of field properties appears where you can define 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 *Administration Guide*.

Use Case Example: Formatting a Date Attribute

The following overview presents logical attribute design-time and run-time activities. The use case is an example of a user account expiration date.

Design Time Activities

Problem

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

Solution

Create a logical attribute 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 for handling expiration dates and other dates

1. Register the new logical attribute handler in the Management Console:
2. Create the logical attribute handler DateFormatAdapter by extending the class LogicalAttributeAdapter. The custom handler should implement 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 Management Console to 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, as follows:

```
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.getLogicalAttributeName(
                        LOGICAL_ATTRIBUTE_MONTH);
    String _day = attrContext.getLogicalAttributeName(
                        LOGICAL_ATTRIBUTE_DAY);
    String _year = attrContext.getLogicalAttributeName(
                        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 CA Identity Manager environment.
5. When creating the associated task screen in the User Console, assign the three logical attribute names to the Month, Day, and Year fields you add to 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, CA Identity Manager retrieves the value of the Date physical attributes (mm.dd.yyyy) from the database, and then calls toLogical() in the handler.
2. The toLogical() method's custom code converts the physical attribute value to the three logical attribute values. CA Identity Manager then displays the logical attribute values in the associated Month, Day, and Year fields.

3. While working in the task screen, the end user decides to edit an account, which is about to expire. To do so, the user changes the expiration date, and then submits the task screen.
4. CA 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. CA Identity Manager then writes the Date value in the data store.

Screen-Defined Logical Attributes

Screen-defined logical attributes are Profile tab screen fields in 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 CA Identity Manager administrators to define and collect information locally, and not persist the data 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 the Configure Standard Profile Screen, which is 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 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 edit button next to the Profile tab.

   The Configure Profile screen appears.

4. Click the browse button to the right of the Screen field.

   The Select Screen Definition list appears.

5. Click the option 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 edit button next to the field you want to modify.
   - Click an Add button to define a new field.

Note: For more information about the Configure Standard Profile Screen, see the Online Help.

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 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, because 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 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;
}
```

Logical Attribute Handler: Option List

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 CA Identity Manager features:

Logical Attribute API

Populate the option list, convert values between the associated logical attribute and physical attribute, and validate the selected values.

Management Console

Configure the logical attribute handler that populates the option list and processes the selected values.

User Console

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, items in a data source, or items in a text file.

CA 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 Management Console. When you do so, be sure to 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 can be associated with one or more physical attributes, and it can 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, similar to a text field or other field type. You add the option list when you are defining a task screen in the User Console.

The only attribute you need to add to the task screen is the logical attribute that displays the selected values. CA 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. Be sure that the field to be configured with the logical attribute is included on the task screen.
2. Click on the edit button for the field.
3. Select the name of the logical attribute that displays 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 drop-down list.

More Information:

Example: Assigning a Logical Attribute to a Field (see page 53)

Option List Configuration Summary

The following table summarizes the basic configuration properties for logical and physical attributes associated with an option list. Set the properties in the User Console and the Management Console:

Attribute Type	Purpose	User Console Style Type	Management Console Properties
Logical attribute	Contains one selected value	Single-Select or Dropdown	Name and Attribute Name
Logical attribute	Contains one or more selected values	Multi-Select	Name, Attribute Name, and Multi-valued
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, CA Identity Manager does the following:

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

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

The sample is installed in the following default directory:
admin_tools\samples\LogicalAttributes\StateSelector

Logical Attribute Handler: Self-Registration

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.

Be sure that you are familiar with configuring logical attributes before reading the following sections.

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

More Information:

Logical Attribute Handler Configuration (see page 52)

How a Self-Registration Handler Works

A self-registration handler is similar to a typical logical attribute handler in the following ways:

- You use the Logical Attribute API to write the custom handler. The custom handler extends the OrgSelectorAdapter abstract class, which extends LogicalAttributeAdapter (javadoc-im/com/netegrity/imapi/LogicalAttributeAdapter.html).
- You define an Organization Selector (self-registration handler) in the Management Console for your CA 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 Management Console to your self-registration screen.
- CA 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 CA 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 CA Identity Manager.

Three differences between typical logical attribute handlers and self-registration handlers are as follows:

- The OrgSelectorAdapter class is only used with self-registration handlers.
- CA 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 (javadoc-im/com/netegrity/imapi/LogicalAttributeAdapter.html).

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

- 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 CA Identity Manager.

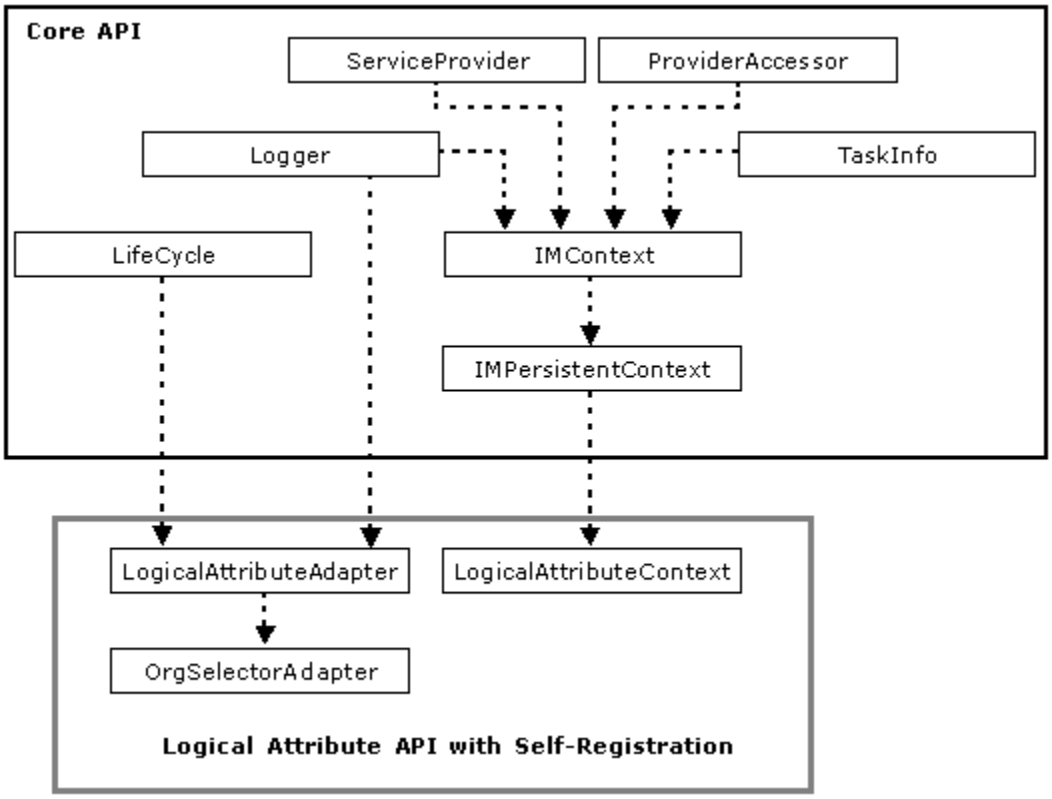
Note: Because OrgSelectorAdapter extends the abstract class LogicalAttributeAdapter, your derived class must declare the methods in LogicalAttributeAdapter even if the methods remain empty.

More Information:

Logical Attribute API Summary (see page 46)

Self-Registration and the API Functional Model

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



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 Management Console.

To define a self-registration handler

1. In the Management Console, click 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 do the following:

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

For detailed instructions, click Help on the Organization Selector screen.

Note: There can be only one self-registration handler per CA 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 following default directory:

admin_tools\samples\LogicalAttributes\OrgSelector

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

Logical Attribute Handler: Forgotten Password

CA 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 select questions from a predefined list.

When you initiate the Forgotten Password task, CA Identity Manager does the following:

1. 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, CA Identity Manager can issue a temporary password (which is typically displayed), or it 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 more information about how the Forgotten Password task is configured, see the *Configuration Guide*.

For example, you may have a setup where your questions and answers are stored in a database and you do not want to update a CA 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, a separate attribute is required 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 Management Console.

To define a Forgotten Password Handler

1. In the Management Console, click 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, click Help on the Forgotten Password screen.

Chapter 4: Business Logic Task Handler API

This section contains the following topics:

Business Logic Task Handler API Overview (see page 69)
Configuring Business Logic Task Handlers (see page 73)
API Components (see page 75)
Calling Sequence (see page 77)
Custom Messages (see page 78)
Access to Managed Objects (see page 80)
JavaScript Business Logic Task Handlers (see page 81)
Default Installed Handlers (see page 83)

Business Logic Task Handler API Overview

The Business Logic Task Handler API (javadoc-im/com/netegrity/imapi/BLTHAdapter.html) 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).

A business logic task handler (BLTH) can be implemented in Java or server-side JavaScript.

Business Logic Task Handler API Summary

Operations

Task-Level Validation

Called by

CA 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
- Before submitting the task to the BLTH adapter.

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

Operates on

- CA Identity Manager objects associated with any task screen in the CA Identity Manager environment. Performed by global handlers. (Java implementations only.)
- CA 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

Updates happen in the following ways:

- In the task session, through the get... methods in BLTHContext. Because CA 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 CA Identity Manager objects, and information in third-party data sources.

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

Task Sessions

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 (javadoc-im/com/netegrity/imapi/BLTHContext.html) object passed to business logic task handlers.

A task session includes the following:

- The task context, that is, the task being performed, the administrator who is performing the task, and the CA Identity Manager managed object affected by the task.

  This *managed object* is also 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 150)

Task Session Management

CA Identity Manager manages the task session.

A task session begins when an administrator selects a task to perform in the 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 CA Identity Manager creates events for the task, errors can be immediately presented to the administrator performing the task.

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, CA Identity Manager has already performed the search.

More Information:

Access to Managed Objects (see page 80)

Use Case Examples

The following are 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

  CA Identity Manager is shipped with a predefined business logic task handler that checks for possible duplication of objects before 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 has 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, CA 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.

Scope of Business Logic Task Handlers

A business logic task handler can be either task-specific or global in scope. These types of business logic task handlers are described as follows:

- *Task-specific* business logic task handlers apply to a particular task or set of tasks. The code is specified directly in the User Console 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 CA Identity Manager environment. They are useful for corporate policies that must 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:

Configuring Business Logic Task Handlers (see page 73)

Configuring Business Logic Task Handlers

Business logic task handlers are configured in the following locations, depending on scope:

- Global business logic task handlers are configured in the Management Console.
- Task-specific business logic task handlers are configured in the User Console task screen where the handler is used.

More Information:

Scope of Business Logic Task Handlers (see page 73)
How to Configure a Task-Specific BLTH (see page 74)
How to Configure a Global BLTH (see page 74)

How to Configure a Global BLTH

Define the fully qualified class name and other properties of a global business logic task handler (BLTH) in the Management Console.

To configure a global business logic task handler

1. In the Management Console, click Environments.
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 CA Identity Manager and any custom handlers defined at your site. CA 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 CA Identity Manager
- Modify an existing handler configuration
- Delete a handler
- Change the order of handler execution

Note: For detailed instructions, click help on the Business Logic Task Handlers screen.

How to Configure a Task-Specific BLTH

Define the fully qualified class name and other properties of a a task-specific business logic task handler (BLTH) in the User Console.

To define a task-specific business logic task handler

1. In the User Console, navigate 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. CA 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.

Note: For detailed instructions, click Help on the Business Logic Task Handler Detail screen.

API Components

The following components are in the package com.netegrity.imapi:

BLTHAdapter (javadoc-im/com/netegrity/imapi/BLTHAdapter.html)

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 (javadoc-im/com/netegrity/imapi/BLTHContext.html)

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 may be assigned to

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

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

Extends IMPersistentContext.

IMPersistentContext (javadoc-im/com/netegrity/imapi/IMPersistentContext.html)

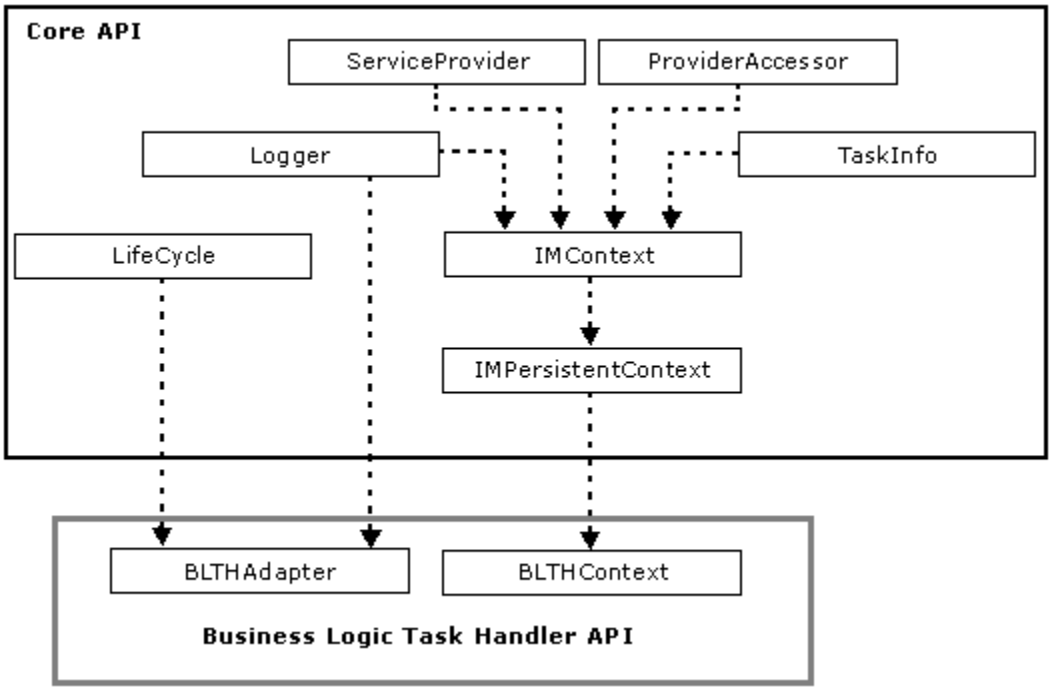
This interface is part of the core API functional model.

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.

Extends IMContext.

Business Logic Task Handler API Model

The components of the Business Logic Task Handler API are part of the CA Identity Manager API functional model, as shown in this diagram:



Sample Business Logic Task Handlers

Sample implementations of business logic task handlers are in the samples\BLTH subdirectory of your CA Identity Manager installation, for example:

admin_tools\samples\BLTH

Java and JavaScript samples are included.

Note: For information about using the samples, see the accompanying readme.txt files.

Calling Sequence

CA 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 the following order:

1. A CA Identity Manager admin task is submitted on a task screen.
2. CA Identity Manager creates a BLTHContext (javadoc-im/com/netegrity/imapi/BLTHContext.html) 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, CA Identity Manager invokes the first handler referenced on the task screen.
4. After all task-specific handlers are executed, CA Identity Manager invokes any global task handlers defined in the 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 (javadoc-im/com/netegrity/imapi/BLTHAdapter.html). You can implement your adapter so that CA Identity Manager can execute the business logic at various points during the synchronous phase of task execution.

You can implement any 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 before the user submits the task, implement handleValidationBeforeSubmission().
- 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

CA 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:

Example:

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

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
            IMSException imsEx = new IMSException();
            imsEx.addUserMessage("Failed to get administrator unique name");
            throw imsEx;
        }
```

Custom Messages

After a user submits a task, CA 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 is 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() (javadoc-im/com/netegrity/imapi/TaskInfo.html) method inherited by BLTHContext (javadoc-im/com/netegrity/imapi/BLTHContext.html).

Exception Messages

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

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

CA 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 CA Identity Manager selects the appropriate resource bundle from which to retrieve exception messages.

Note: For detailed information see the IMSException (javadoc-im/com/netegrity/ims/exception/IMSException.html) class in the Javadoc Reference.

More Information:

IMSException (see page 161)

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 CA Identity Manager to display the message on the screen:

Example:

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

Access to Managed Objects

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

- Using the CA Identity Manager provider objects (javadoc-im/com/netegrity/llsdk6/imsapi/provider/package-summary.html) 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 CA Identity Manager events are generated, and no workflow approvals, auditing, or security checks are performed.

- Using the CA Identity Manager BLTHContext (javadoc-im/com/netegrity/imapi/BLTHContext.html) 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. CA Identity Manager generates events for the task, and workflow approvals, auditing, and security checks can be performed.

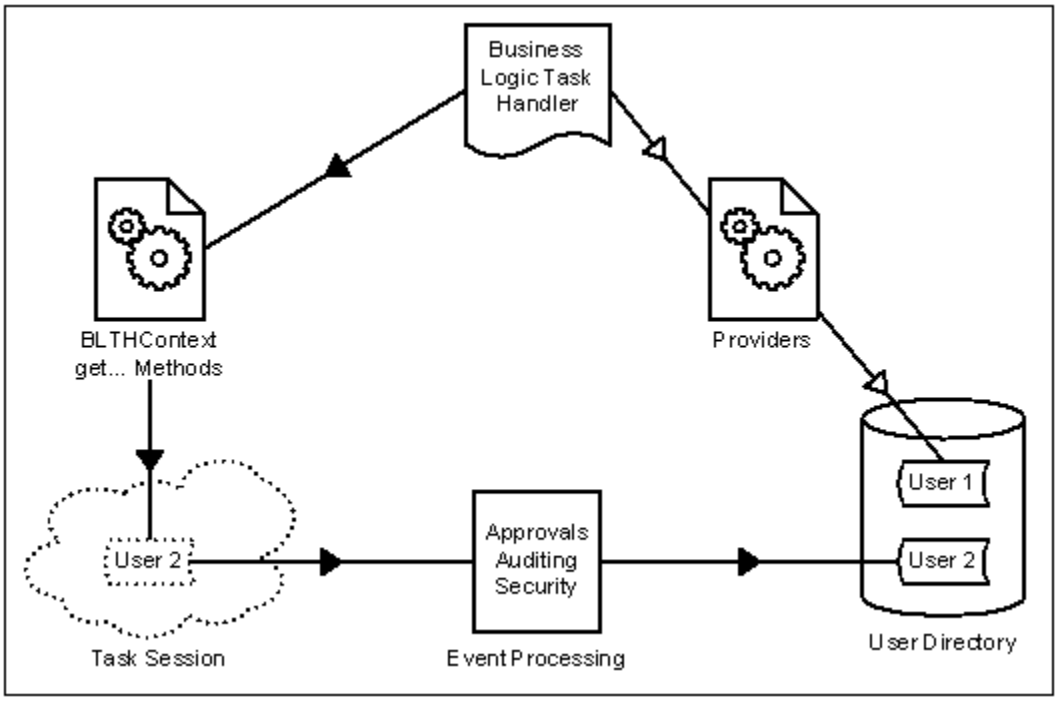
  The changes are not committed to the data store until all events associated with the task have been executed.

More Information:

Access to Objects in the Data Store (see page 146)
Access to Objects in a Task (see page 150)

Managed Object Process Diagram

In the following 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, CA Identity Manager commits User 2 to the data store.



JavaScript Business Logic Task Handlers

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 CA 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 variable in all other class references where necessary.

JavaScript Method Reference

A JavaScript handler implements the same methods as the Java BLTHAdapter (javadoc-im/com/netegrity/imapi/BLTHAdapter.html) 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 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
}
```

Default Installed Handlers

CA Identity Manager has the following two business logic task handlers which are installed with the product by default:

BlthPasswordServices

Validates a new or changed password before it is submitted. This handler validates against any password policies and displays the appropriate error. It is for both CA SiteMinder and CA Identity Manager password policies.

BlthCheckForDuplicates

Checks to see if an organizaton or group object already exists before it is submitted.

By default, CA Identity Manager does not allow duplicate names of organizations and groups. However, the handler has the following attributes for allowig duplicate names:

checkDuplicateGroup

Allows the creation of groups with the same name under different organizations. The default value is true. If false, CA Identity Manager does not check for duplicate group names.

checkDuplicateOrg

Allows the creation of organizations with the same name. The default value is true. If false, CA Identity Manager does not check for duplicate organization names.

Check for Preventative Identity Policy

Audits preventative identity policy violations (as defined by the the Action on Apply Policy settings for an identity policy) before the task is submitted.

Perform Analytics for Bulk Loader Task

Runs Smart Provisioning logic on bulk load tasks when CA Identity Governance is integrated with CA Identity Manager.

Provisioning Role Handler

Creates a provisioning role object in the object store when a provisioning role is created.

Check for Rename Provisioning Role

Monitors Modify Provisioning Role tasks for changes to the friendly name of provisioning roles. If the role name change is detected, the warning message is displayed.

Renaming a provisioning role can cause unexpected results in account management and compliance operations.

PX Listener

Listens for tasks with Policy Xpress policies that pass policy steps to the Policy Xpress engine.

Chapter 5: Event Listener API

This section contains the following topics:

Tasks and Events (see page 85)
Event Listener API Overview (see page 86)
Event Listener Execution (see page 90)
Access to Managed Objects (see page 94)
Event Generation (see page 95)
Event Listener Configuration (see page 95)

Tasks and Events

A CA Identity Manager task is made up of one or more events. When CA Identity Manager executes a task, it breaks down the task 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
- One or more AddToGroupEvent events, to add the new user to a group
- One or more AssignAccessRoleEvent events, to assign the new user to a role
- 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, specifically, during a particular state of an event.

Event Listener API 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 complete.

More Information:

Event Listener State Diagram (see page 92)

Event Listener Operations

The following list summarizes Event Listener API operations:

Perform Pre-Approval Attribute Updates

- Called by CA 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 CA 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

The Post-Execution operations are called in the following two states:

Post-Successful-Execution

- Called by CA Identity Manager after the event is executed
- Operates on events
- Method called: after()
- Object update only through the providers
- Cannot validate

Post-Failed-Execution

Called by CA Identity Manager after the event failed to be executed

Operates on events

Method called: failed()

Object update only through the providers

Cannot validate

Use Case Examples

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, CA 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 CA Identity Manager environment. CA 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, CA 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 must be updated. After the event is executed, CA 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 CA 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, CA 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 is configurable through the properties of the CA 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:

EventListenerAdapter (javadoc-im/com/netegrity/imapi/EventListenerAdapter.html)

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. CA Identity Manager calls a different EventListenerAdapter method for each event state and passes EventContext to the method.

Implements LifeCycle and Logger.

EventContext (javadoc-im/com/netegrity/imapi/EventContext.html)

Interface that provides read/write access to managed objects in CA 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 CA 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 (javadoc-im/com/netegrity/imapi/IMPersistentContext.html)

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 interf (javadoc-im/com/netegrity/imapi/EventListenerAdapter.html)ace is part of the core API functional model.

Extends IMContext.

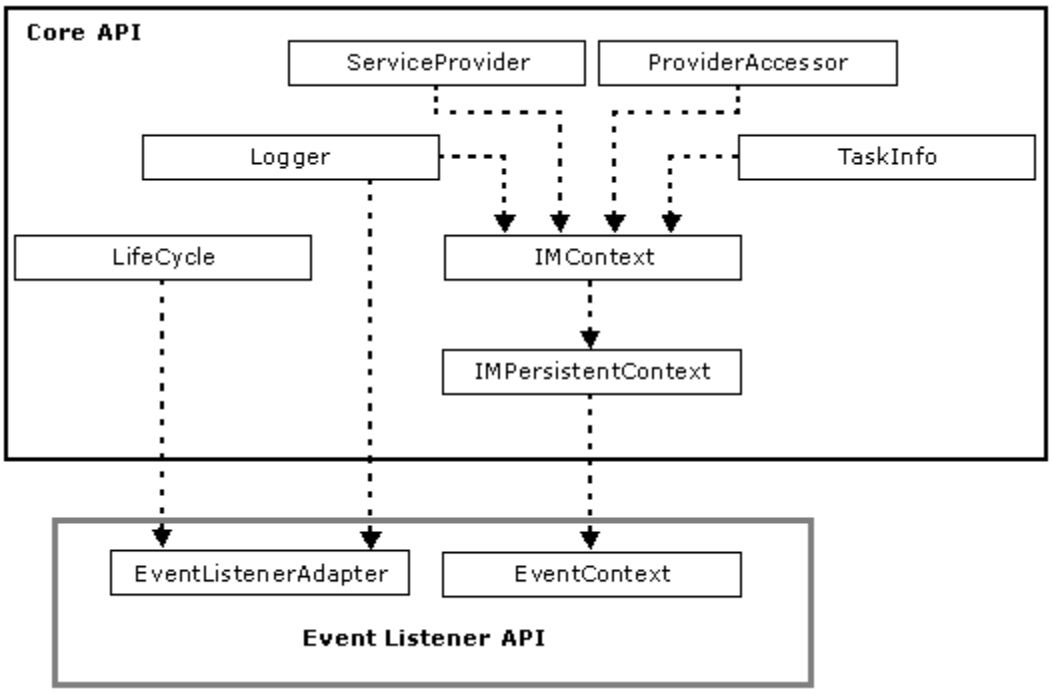
The preceding components are in the package com.netegrity.imapi.

More Information:

Access to Event Objects (see page 153)
Event Listener States (see page 91)

Event Listener API Model

The components of the Event Listener API are part of the CA Identity Manager API functional model, as shown in the following diagram:



Event Listener Execution

CA 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, map the event listener to one or more events. You do so in the Event Listener Properties screen of the Management Console.

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

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

More Information:

Event Listener Configuration (see page 95)

Event Listener States

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

1. Pre-approval state.

   This state occurs at the following times:

   - 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()

2. 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.

     Entry Point in EventListenerAdapter: 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: rejected()

3. Post-execution states.

   - **Post-successful execution**—occurs after the event is executed successfully. Provisioning (automatic, rule-based role and group assignments) are typically made during this state.

     Entry Point in EventListenerAdapter: after()

   - **Post-failed execution**—occurs after the event failed during execution. Provisioning (automatic, rule-based role and group assignments) are typically made during this state.

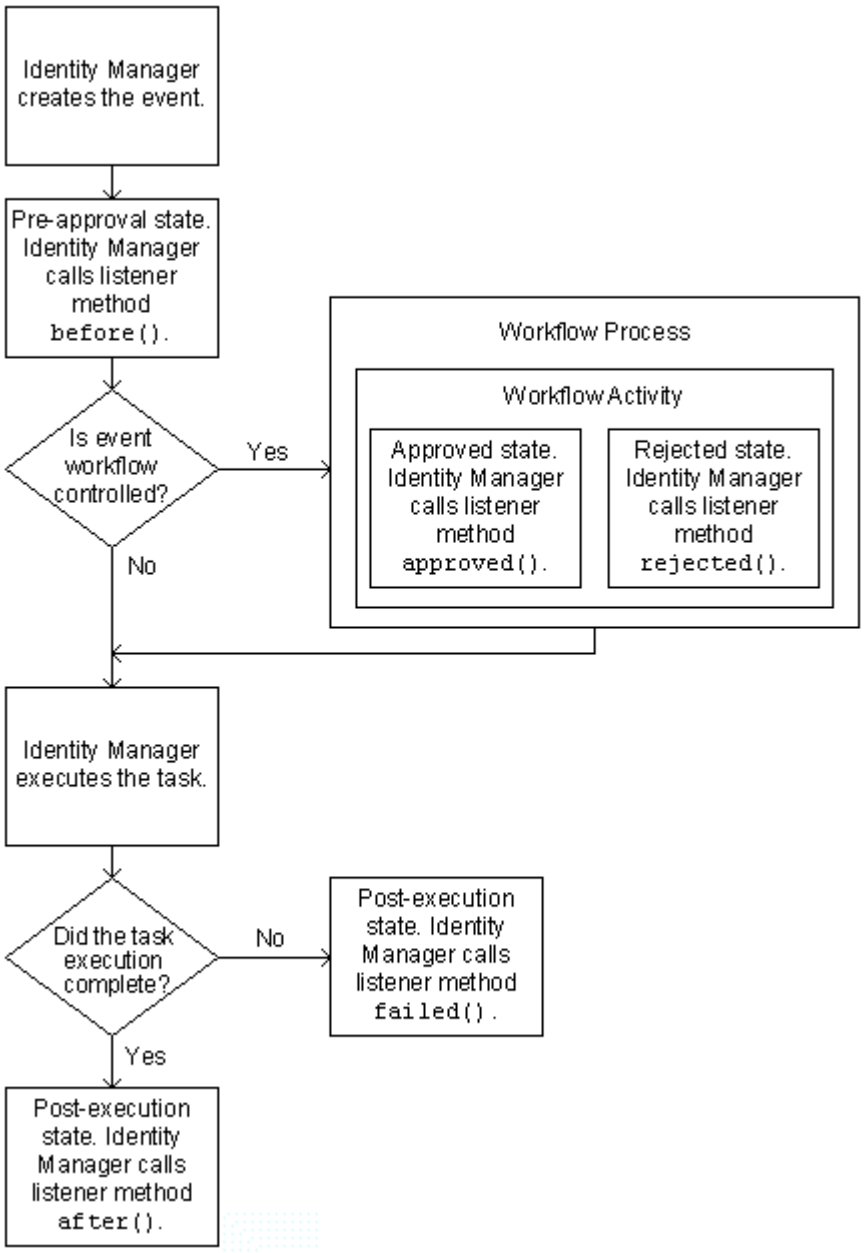
     Entry Point in EventListenerAdapter: failed()

Note: CA Identity Manager View and Audit events do not include these states. Event listeners are not supported for View and Audit events.

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, CA 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 in the samples\EventAdapter subdirectory of your CA Identity Manager installation:

admin_tools\samples\EventAdapter

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

Order of Execution

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

CA 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 Management Console, with the topmost event listener being evaluated first.
- If an event listener is mapped to the current event, CA 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, CA 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, CA Identity Manager advances the event to the next sequential state, and evaluates the next listener according to the rules listed previously.
- If an event listener returns BREAK, no more event listeners are processed for the current event state. CA Identity Manager advances the event to the next sequential state, and evaluates the next listener according to the previous rules.

The CONTINUE and BREAK return codes cause CA 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, CA Identity Manager does not complete the execution of the event.

More Information:

Event Listener State Diagram (see page 92)

Access to Managed Objects

An event listener has access to the following managed objects:

Managed objects in an event

Each CA Identity Manager event is represented by an event object. CA 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 CA Identity Manager passes into the listener.

When 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 CA 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 CA Identity Manager events are generated for the changes, and no workflow approvals, auditing, or security checks are performed.

More Information:

Access to Objects in the Data Store (see page 146)
Retrieving Event Objects (see page 154)
Base Event Interfaces (see page 154)

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. CA 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 the following:

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

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

Event Listener Configuration

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

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

To set event listener properties

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

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

More Information:

Compiling and Deploying (see page 173)

Chapter 6: Validation Rules

This section contains the following topics:

Validation Rules Introduction (see page 97)
About Validation Rules (see page 97)
Using Default Validation Rules (see page 101)
How to Implement Custom Validation Rules (see page 104)
How to Configure Validation Rules (see page 112)
How to Initiate Validation (see page 118)
Sample Implementations (see page 119)

Validation Rules Introduction

Values are assigned to data store attributes through task screen fields or programmatically. Attribute validation rules help ensure that the values users type in task screen fields or that are supplied programmatically meet certain requirements, as in the following examples:

- User directory requirements, such as enforcing a data type, or verifying that an entry such as a date is formatted in a particular way.
- Data integrity. Does an entry make sense in the context of other information about the task screen or according to site-specific business rules?

A validation rule can be directly associated with a task screen field, or be indirectly associated with the field by being associated with a managed object attribute that is configured for the field.

All validation rules directly or indirectly associated with a task screen's fields must be satisfied before CA Identity Manager can begin processing the task. When a supplied value is invalid, a message associated with the violated rule is displayed, and the user can then correct the entry and resubmit the task.

About Validation Rules

Validation rules enforce requirements, such as in the following examples:

- A Quantity field must contain only numeric characters.
- A Telephone Number field must be formatted as nnn-nnn-nnnn.
- An Employee ID field must contain a number no higher than 9999.

- The value typed in a ZIP Code field must be appropriate for the values typed in the City and State field.
- Does the value typed in a Title field qualify the user for the security clearance typed in Security Level?

In addition to verifying a user entry, a validation rule can *change* an entry so that the entry conforms to the rule's requirements without further user intervention, as in the following examples:

- A validation rule for a Telephone Number field requires that telephone numbers be formatted as nnn-nnn-nnnn. If a user types the value 9785551234, the validation rule automatically changes the entry to the correct format, 978-555-1234.
- A validation rule for a Department Number field requires that the number must be prefixed with a three-character code representing the name typed in the Region field. When the prefix is missing or incorrect, the validation rule supplies the correct prefix.

Changing an entry through a validation rule is named *transformation*.

Types of Validation Rules

The two types of validation rules are as follows:

- **Task-level validation**—validates an attribute value against other attributes in the task. For example, you can verify that the area code in a user-supplied telephone number is appropriate for the user's city and state.

  During task-screen configuration, task-level validation rules are directly associated with task screen fields.

  You can use this type of validation to enforce data integrity.

- **Directory-level validation**—validates the attribute value itself, and not in the context of other attributes in the task. For example, you can verify that a user-supplied telephone number matches the nnn-nnn-nnnn format used in the directory.

  In directory.xml, directory-level validation rules are mapped to a managed object attribute through a rule set. The rules in the rule set are applied to any task screen field configured with the attribute.

  You can use this type of validation to enforce user directory requirements.

CA Identity Manager executes task-level validation rules before directory-level validation rules.

Example: Comparing Directory-Level Validation and Task-Level Validation

In this example, a telephone attribute is mapped in directory.xml to a directory-level validation rule requiring telephone numbers to be formatted as nnn-nnn-nnnn. All fields configured with the telephone attribute are validated against the nnn-nnn-nnnn format whether the field appears in a Create User task screen, a Create Supplier task screen, or any other task screen.

If a Telephone Number field appears on a Create Customer task screen, like telephone number fields in other task screens, this field is configured with the telephone attribute that requires the nnn-nnn-nnnn telephone number format. However, because some of the company's customers are located in other states, the Telephone Number field on the Create Customer task screen is also associated with the following task-level validation logic:

- Check the value in the State field.
- When the customer is located out of state, be sure that the area code of the customer's telephone number is appropriate for the customer's state.

Validation Rule Sets

With directory-level validation, one or more validation rules are assigned to a rule set, and the rule set is associated with a managed object attribute.

Rule sets let you define and apply rules in a granular way, such as in the following examples:

- A rule can be used in different rule sets
- Rules can be executed in different combinations

When a rule in a rule set fails (for example, a Java or JavaScript rule returns False), any exception messages associated with the rule are presented to the user. All validation rules associated with the attribute must be satisfied before the attribute is considered validated.

Order of Execution

Rules are executed in the order in which they are listed in the rule set. CA Identity Manager executes each rule in a rule set separately, and transparently continues to each subsequent rule in the rule set unless a rule fails.

Because validation rules are executed in a predictable order, you can implement rules whose actions are dependent upon the outcome of previous rules, as in the following examples:

- One rule's output can become input to the next rule.
- When a field value is changed during validation, the new value can be evaluated in subsequent rules.

Basics of Validation Rule Definition

Perform the following basic operations when defining custom validation rules:

- **Implement a validation rule**. Implement a validation rule in any of the following ways:
  - Regular expression
  - JavaScript
  - Java class
- **Integrate a validation rule with CA Identity Manager through a task screen or directory.xml**. Do so either inline (directly in the task screen or directory.xml file) or by reference (referencing a JavaScript source file or compiled Java class file), as shown in the following table:

	Inline	By Reference
Regular Expression	directory.xml or task screen	—
JavaScript	directory.xml or task screen	Source file referenced in directory.xml
Java	—	Class file referenced in directory.xml or task screen

- **Associate one or more validation rules with a task screen field**. Do so in either or both of the following ways:
  - With task-level validation, you assign a validation rule directly to a field on a particular task screen.

    Task-level validation has task-specific scope—that is, it can be used only in the context of the particular task screen where it is assigned.
  - With directory-level validation, you map a rule set to a managed object attribute in directory.xml. Any task screen field that is configured with the attribute is validated against the rules in the rule set.

    Directory-level validation has global scope. This means that directory-level validation can be used on any field configured with the managed object attribute, regardless of the task screen that contains the field, and regardless of the CA Identity Manager environment that includes the task screen.

Using Default Validation Rules

CA Identity Manager is shipped with the following types of default validation rules:

- Data validation of task screen fields
- Predefined validation rules defined in the directory.xml file

Default Data Validations

By default, CA Identity Manager checks certain data when an administrator submits a task for processing. When the data is invalid, CA Identity Manager stops processing the task and displays an error message. The data validations that CA Identity Manager performs are based on the type of task, as shown in the following table:

Tasks	Validation
All tasks	Required fields must have a value.
Create User Create Group Create Organization Create Access Role Create Access Task Create Admin Role Create Admin Task	An administrator cannot create an object with the same name as an existing object of the same type. For example, an administrator cannot create two admin roles with the same name. **Note:** For users and groups, CA Identity Manager checks only the current organization.

Tasks	Validation
Create User Create Group Create Organization	An administrator cannot create a user, group, or organization with a name that contains any of the following characters: ■ comma (,) ■ single quote (’) ■ double quote (") ■ asterisk (*) ■ ampersand (&) ■ slash (/) ■ back slash (\) ■ less than sign (<) ■ greater than sign (>) ■ equal to sign (=) ■ plus sign (+) ■ semicolon (;) ■ pound sign (#) ■ leading or trailing spaces **Note:** Organization names can contain a comma (,) or an ampersand (&).
All Create and Modify tasks	Attributes with read/write permission (excluding passwords) cannot contain the following characters: ■ comma (,) ■ percent sign (%) ■ less than sign (<) ■ greater than sign (>) ■ semicolon (;) These characters are vulnerable to cross-site scripting attacks.

Tasks	Validation
Create User Self-register Change My Password Reset User Password Any custom task that collects and stores user passwords	If you are using CA SiteMinder®'s Password Services feature to enforce password rules (such as minimum length), user passwords are validated against these rules. If the password does not satisfy the password policy, the password is not accepted. **Note:** For more information, see the *CA SiteMinder® Web Access Manager Policy Server Configuration Guide*.
Modify User	Administrators cannot give themselves a role or the ability to assign a role.
Forgotten Password	If a user profile does not have a password hint and answer, that user cannot use the forgotten password feature.
Delete User Enable/Disable User	Administrators cannot delete their own profile or change the status of their account.
Delete Organization	Administrators cannot delete the organization where they are assigned the role that contains the Delete Organization task. Consider an administrator who is assigned the Organization Manager role in the Dealers organization. The Organization Manager role enables this user to delete organizations. This administrator can delete suborganizations of Dealers, but cannot delete Dealers.
Modify Organization	Administrators cannot modify the organization where they are assigned the role that contains the Modify Organization task.

Predefined Validation Rules

CA Identity Manager includes the following validation rules predefined in the directory.xml file. Predefined validation rules are used for directory-level validation only, as shown in the following table:

Predefined Rule Name	Description
Phone pattern	Enforces the following format for telephone numbers: +nn nnn-nnn-nnnn
Set international	Adds the prefix +1 to an international telephone number.

Predefined Rule Name	Description
Valid User	Verifies that the specified User object exists in the directory.
Valid Group	Verifies that the specified Group object exists in the directory.
Valid Organization	Verifies that the specified Organization object exists in the directory.

Predefined validation rules and custom validation rules can appear in the same rule set.

How to Implement Custom Validation Rules

You can implement a validation rule for one of the following:

- Regular expression
- JavaScript
- Java class

Regular Expression Implementation

A validation rule can be based on regular expression pattern matching. For example, you can do the following:

- Specify a list of invalid characters or values for an attribute
- Restrict the user from typing invalid constructs, such as an improperly formed DN or telephone number

The following JavaScript example enforces telephone number format as +nn nnn-nnn-nnnn:

```
phone=/\+\d{1,3} \d{3}-\d{3}-\d{4}/;
```

Wrap regular expressions defined in XML in CDATA, as in the following example:

```
<ValidationRule name="Phone pattern" description="+nn nnn-nnn-nnnn"
 messageid="4001">
   <RegularExpression>
      <![CDATA[ ((\+|\d)*+(\s*|\x2D))?\d\d\d-\d\d\d-\d\d\d\d]]>
   </RegularExpression>
</ValidationRule>
```

Validation rules based on regular expressions must comply with the requirements defined in the java.util.regex package.

JavaScript Implementation

A JavaScript-based validation rule must implement the relevant interface, depending on whether the rule is used for task-level validation or directory-level validation.

At validation time, CA Identity Manager calls validate() and passes the value to be validated.

JavaScript Interface for Task-Level Validation

The definition of the JavaScript interface for task-level validation is as follows:

Syntax

```
public boolean validate(
   BLTHContext context,
   String attributeValue,
   StringRef changedValue,
   StringRef errorMessage
   );
```

Parameters

context

Input parameter

Specifies an object that contains methods for retrieving information in the current task session.

attributeValue

Input parameter

Specifies the value of the attribute being validated.

changedValue

Output parameter

Provides an optional transformation value that replaces the user-supplied value being validated. If no transformation is necessary, pass back null.

errorMessage

Output parameter

If validation fails, it displays a message to the user.

The message is displayed through AttributeValidationException. If the method returns false, CA Identity Manager generates this exception.

Comments

The output parameters *changedValue* and *errorMessage* are of data type StringRef. StringRef is a predefined data type that contains the field *reference* to which you assign a value, as shown in the following examples:

- Add a 1 prefix for a properly formatted telephone number:
  ```
  changedValue.reference="+1 " + phoneNumber;
  ```
- Provide an error message for an improperly formatted number:
  ```
  errorMessage.reference="Phone number " + phoneNumber +
              " does not match the format nnn-nnn-nnnn.";
  ```

Returns

- True. The implementation considers the value in *attributeValue* to be valid, or it passes back a transformed value in *changedValue*.
- False. The implementation considers *attributeValue* to be invalid. CA Identity Manager generates an AttributeValidationException that includes *errorMessage*.

JavaScript Interface for Directory-Level Validation

The definition of the JavaScript interface for directory-level validation is as follows:

Syntax

```
public boolean validate(
   String attributeValue,
   StringRef changedValue,
   StringRef errorMessage
   );
```

Parameters

attributeValue

Input parameter

Specifies the value of the attribute being validated.

changedValue

Output parameter

Provides an optional transformation value that replaces the user-supplied value being validated. If no transformation is necessary, pass back null.

errorMessage

Output parameter

If validation fails, it displays a message to the user.

The message is displayed through AttributeValidationException. If the method returns false, CA Identity Manager generates this exception.

Comments

The output parameters *changedValue* and *errorMessage* are of data type StringRef. StringRef is a predefined data type that contains the field *reference,* to which you assign a value, as shown in the following examples:

- Add a 1 prefix for a properly formatted telephone number:
  ```
  changedValue.reference="+1 " + phoneNumber;
  ```
- Provide an error message for an improperly formatted number:
  ```
  errorMessage.reference="Phone number " + phoneNumber +
              " does not match the format nnn-nnn-nnnn.";
  ```

Returns

- True—the implementation considers the value in *attributeValue* to be valid, or it passes back a transformed value in *changedValue*.
- False—the implementation considers *attributeValue* to be invalid. CA Identity Manager generates an AttributeValidationException that includes *errorMessage*.

Java Implementation

A Java-based validation rule must implement the relevant interface, depending on whether the rule is used for task-level validation or directory-level validation.

At validation time, CA Identity Manager calls validate() and passes the value to be validated.

Java Interface for Task-Level Validation

The definition of the Java interface for task-level validation is as follows:

Syntax

```
public interface TaskValidator {
public class StringRef {
   public String reference = new String();
   public String toString(){return reference;}
public boolean validate(
   BLTHContext ctx,
   String attrValue,
   StringRef updatedValue,
   StringRef errorMessage
   ) throws AttributeValidationException;
}
```

Parameters

ctx

Input parameter

Specifies an object that contains methods for retrieving information in the current task session.

attrValue

Input parameter

Specifies the value of the attribute being validated.

updatedValue

Output parameter

Provides an optional transformation value that replaces the user-supplied value being validated. When no transformation is necessary, pass back null.

errorMessage

Output Parameter

If validation fails, it displays a message to the user.

Comments

For more information about Java validation rules and on managed objects, see the CA Identity Manager Javadoc.

Returns

- True—the implementation considers the value in *attributeValue* to be valid, or it passes back a transformed value in *changedValue*.
- False—the implementation considers *attributeValue* to be invalid.

Throws

AttributeValidationException

Java Interface for Directory-Level Validation

The definition of the Java interface for directory-level validation is as follows:

Syntax

```
public interface IAttributeValidator {
public class StringRef {
   public String reference = new String();
   public String toString(){return reference;}
public boolean validate(
   Object attributeValue,
   StringRef changedValue,
   StringRef errorMessage
   ) throws AttributeValidationException;
}
```

Parameters

attributeValue

Input parameter

Specifies the value of the attribute being validated.

changedValue

Output parameter

Provides an optional transformation value that replaces the user-supplied value being validated. When no transformation is necessary, pass back null.

errorMessage

Output parameter

If validation fails, it displays a message to the user.

Comments

If the validation operation requires managed objects from the directory, use AttributeValidator. This abstract class implements the IAttributeValidator interface, and includes a method for retrieving the managed object providers.

Returns

- True—the implementation considers the value in *attributeValue* to be valid, or it passes back a transformed value in *changedValue*.
- False—the implementation considers *attributeValue* to be invalid.

Throws

AttributeValidationException.

Exceptions

AttributeValidationException is thrown when a validation rule cannot validate an attribute value supplied in a task screen field or programmatically. The exception contains one or more messages that are presented to the user, enabling the user to correct the entry and resubmit the task.

How this exception is thrown and how the error messages are presented for the exception depends on whether the rule is implemented as JavaScript, a Java class, or a regular expression.

Exceptions with Task-Level Validation

With task-level validation errors, AttributeValidationException is thrown as shown in the following table:

Rule Type	How Thrown	Error Message Source
Regular expression	By CA Identity Manager if the regular expression validation fails.	CA Identity Manager uses a generalized exception message.
JavaScript	By CA Identity Manager if the validate() method returns False.	The *errorMessage* parameter of the validate() method.
Java	By the custom validation rule or by CA Identity Manager. CA Identity Manager throws the exception when the custom rule does not and the custom rule's validate() method returns False.	One of the following sources: ■ If the custom validation rule throws the exception, the exception's constructor. The constructor lets you specify the ID of a message in a resource bundle and the text of an additional message. ■ If CA Identity Manager throws the exception, the *errorMessage* parameter of the validate() method.

If the validation rule implementation does not provide an error message, CA Identity Manager uses a generalized error message.

Exceptions with Directory-Level Validation

Exception messages for directory-level validation errors come from two sources:

- A resource bundle. In directory.xml, definitions of all types of validation rules (Java, JavaScript, and regular expression) include the attribute messageid. This ID maps to a custom exception message in the resource bundle IMSExceptions.properties. When AttributeValidationException is thrown, CA Identity Manager includes the mapped message with other error information that may be defined for the validation rule.
- Custom validation rule code. Java and JavaScript implementations can define additional exception messages for the rule. If a validation error occurs in the Java or JavaScript rule, the message is presented to the user with the message that is mapped to the rule in the resource bundle.

  The sources of these Java and JavaScript exception messages are defined in the previous table.

  This feature does not apply to directory-level validation rules implemented as regular expressions.

Note: For more information about exception messages in resource bundles, see AttributeValidationException in the CA Identity Manager Javadoc.

AttributeValidationException Constructor

When you create an AttributeValidationException object for a Java validate() method, use the following constructor:

Syntax

```
public AttributeValidationException(String attrName,
   String attrValue,
   String messageid,
   String message);
```

Parameters

attrName

Specifies the name of the managed object attribute being validated.

attrValue

Specifies the value to validate.

messageid

If the value cannot be validated, it provides the ID associated with the message to display. The ID corresponds to a message in the resource bundle IMSExceptions.properties.

message

Provides an additional message that can be displayed to the user. This parameter gives you an opportunity to display a more specific message than the one in the resource bundle, or a message from a custom resource bundle.

Note: For more information about AttributeValidationException, see the CA Identity Manager Javadoc.

How to Configure Validation Rules

Configure a validation rule by integrating it with CA Identity Manager, and by directly or indirectly associating it with a task screen field.

How you configure a validation rule determines whether you want the rule applied to a field in a particular task screen (task-level validation) or a field in any task screen (directory-level validation), as follows:

- With task-level validation, you make a direct association between the rule and a field in a particular task screen. Validation is performed on the field in the context of that task screen only.
- With directory-level validation, the association between the rule and the task screen field is indirect, as follows:
  - In directory.xml, you specify the validation rule, add the rule to a rule set, and associate the rule set with a managed object attribute.
  - In the User Console, a field that is configured with the managed object attribute is validated against the rule set mapped to the attribute.

Validation is performed on any field configured with the attribute, regardless of the task screen that contains the field, and regardless of the CA Identity Manager environment that contains the task screen.

How to Configure Task-Level Validation

Configure task-level validation in the User Console, when defining field properties on a profile task screen. The basic steps are as follows:

1. Navigate to the Field properties section of the profile configuration screen containing the field to be validated.

   Note: For more information about field properties, see the *Administration Guide* and the User Console online help.

2. Specify a value in one of the following fields, depending on how the validation rule is to be implemented:

   - Validation Expression. Contains a regular expression that performs the validation.
   - Validation Java Class. Contains the fully qualified name of a Java class that performs the validation, for example:

     ```
     com.mycompany.MyJavaValidator
     ```

     CA Identity Manager expects the class file to be located in the root directory designated for custom Java class files.
   - Validation JavaScript. Contains the complete JavaScript code that performs the validation.

     You must provide JavaScript code in this field. With task-level validation, you cannot reference a file containing JavaScript code.

Note: For information about defining other field properties on a profile configuration screen, click the Help button on the screen.

How to Configure Directory-Level Validation

You configure directory-level validation in the directory.xml file and in a task screen. The basic steps are as follows:

- In the directory.xml file, do the following:
  - Specify a validation rule in the ValidationRule element.
  - Specify a rule set in the ValidationRuleSet element. A rule set contains one or more predefined rules, custom validation rules, or rules of both types.
  - Associate a rule set with a managed object attribute in the ImsManagedObjectAttr element.
- In a task screen, the field to be validated must be configured with the attribute mapped to the rule set.

Integration of Directory-Level Validation with CA Identity Manager

Define validation rules and rule sets to CA Identity Manager through the ImsManagedObjectAttrValidation element of the directory.xml file.

The schema for the ImsManagedObjectAttrValidation element is as follows:

```
<xs:element name="ImsManagedObjectAttrValidation" minOccurs="0">
   <xs:complexType>
      <xs:sequence>
         <xs:element name="ValidationRule" minOccurs="0"
                                             maxOccurs="unbounded">
            <xs:complexType>
               <xs:choice>
                  <xs:element name="Java">
                     <xs:complexType>
                        <xs:attribute name="class" type="xs:string"
                                                   use="required"/>
                     </xs:complexType>
                           </xs:element>
                  <xs:element name="JavaScript">
                     <xs:complexType>
                        <xs:simpleContent>
                           <xs:extension base="xs:string"/>
                        </xs:simpleContent>
                     </xs:complexType>
                  </xs:element>
                  <xs:element name="JavaScriptFile">
                     <xs:complexType>
                      <xs:attribute name="file" type="xs:string"
                                                 use="required"/>
                     </xs:complexType>
                  </xs:element>
                  <xs:element name="RegularExpression">
                     <xs:complexType>
                        <xs:simpleContent>
                           <xs:extension base="xs:string"/>
                        </xs:simpleContent>
                     </xs:complexType>
                  </xs:element>
               </xs:choice>
               <xs:attribute name="name" type="xs:string"
                                                   use="required"/>
               <xs:attribute name="description" type="xs:string"
                                                   use="optional"/>
               <xs:attribute name="messageid" type="xs:string"
                                                   use="required"/>
            </xs:complexType>
         </xs:element>
```

```
        <xs:element name="ValidationRuleSet" minOccurs="0"
                                          maxOccurs="unbounded">
           <xs:complexType>
              <xs:sequence>
                 <xs:element name="ValidationRule"
                                          maxOccurs="unbounded">
                    <xs:complexType>
                       <xs:attribute name="name" type="xs:string"
                                          use="required"/>
                    </xs:complexType>
                 </xs:element>
              </xs:sequence>
              <xs:attribute name="name" type="xs:string"
                                          use="required"/>
              <xs:attribute name="description" type="xs:string"
                                          use="optional"/>
           </xs:complexType>
        </xs:element>
     </xs:sequence>
  </xs:complexType>
</xs:element>
```

The following elements are defined:

ValidationRuleSet

Consists of one or more predefined or custom validation rules. A validation rule is specified in the ValidationRule element.

Both predefined rules and custom rules can appear in the same rule set. Also, a rule set can contain any combination of Java, JavaScript, and regular expression implementations.

Validation rules are performed in the order in which they appear in ValidationRuleSet. This allows for cascading validation, where output from one rule is used as input to the next.

ValidationRuleSet is associated with a managed object attribute in the ImsManagedObjectAttr element of the directory.xml file.

ValidationRule

Specifies a validation rule for use in a ValidationRuleSet.

ValidationRule must contain only *one* of the following subelements:

- Java. References the Java class file that implements the rule.
- JavaScript. Contains the inline JavaScript code that implements the rule.
- JavaScriptFile. References the JavaScript source file that implements the rule.
- RegularExpression. Contains the inline regular expression that implements the rule. The regular expression must be wrapped in CDATA.

Key Attributes

Most of the attributes of the previously described elements are self-explanatory. However, the following attributes require explanation:

- Attribute class of element <Java>

  With Java validation rules, the Java class must be deployed in the following root location within your application server:

  `iam_im.ear\custom`

  Class files in this root location must be fully qualified, but need no other path information, for example, com.mycompany.MyJavaImpl.

- Attribute file of element <JavaScriptFile>

  With a validation rule implemented in a JavaScript source file, the file must be deployed in the following root location within your application server:

  `iam_im.ear\custom\validationscripts`

  JavaScript source files in this root location are referenced by name only, for example, MyJavaScriptImpl.js.

- Attribute messageid of element <ValidationRule>

  The message id specified in this attribute maps to an error message in the resource bundle IMSExceptions.properties.

  All types of validation rules (Java, JavaScript, JavaScriptFile, and RegularExpression) contain a messageid attribute.

Example: Inline Regular Expression

The following example shows the predefined Phone pattern validation rule, which is included in the rule set Phone format. The rule is implemented inline as a regular expression:

```
<ValidationRule name="Phone pattern" description="+nn nnn-nnn-nnnn"
                                                    messageid="4001">
   <RegularExpression>
      <![CDATA[ ((\+|\d)*+(\s*|\x2D))?\d\d\d-\d\d\d-\d\d\d\d]]>
   </RegularExpression>
</ValidationRule>
<ValidationRuleSet name="Phone format" description=
                                "Verify format +nn nnn-nnn-nnnn">
   <ValidationRule name="Phone pattern" />
</ValidationRuleSet>
```

In the preceding example, messageid="4001" maps to the following line in IMSExceptions.properties:

```
4001=Attribute Validation: {0} value must match regular expression
                                                        nnn-nnn-nnnn.
```

Example: Reference to JavaScript File

The following example specifies the rule EndWithZ_js. This rule is implemented in JavaScript, and the script is located in the file EndWithZ.js. The rule set that includes the rule is not shown in the example:

```
<ValidationRule name="EndWithZ_js" messageid="custom-5001">
   <JavaScriptFile file="EndWithZ.js" />
</ValidationRule>
```

In the preceding example, the JavaScript file is assumed to be in the following default location:

iam_im.ear\custom\validationscripts

Association of a Validation Rule Set with a Managed Object Attribute

Associate a validation rule set with a managed object attribute through the ImsManagedObjectAttr element of the directory.xml file.

In the following example, the validation rule set Phone format is associated with the managed object attribute telephonenumber:

```
<ImsManagedObjectAttr physicalname="telephonenumber" displayname="Business Phone"
description="Business Phone" valuetype="String" required="false"
multivalued="false" maxlength="0" validationruleset="Phone format" />
```

Note: When a managed object attribute is associated with a validation rule set, the rule set name is displayed in the Attribute Properties screen of the Management Console.

Association of a Validation Rule Set with a Task Screen Field

With directory-level validation, you can associate a rule set with a task screen field indirectly, as follows:

1. Associate the rule set with a managed object attribute, as described in the previous section.
2. Be sure that the task screen field to be validated is configured with the managed object attribute associated with the rule set. At runtime, a field value supplied by an end user is validated against the rules in the rule set.

Typically, task screen fields are already configured with attributes. However, you can add a field to a task screen, or you can change the attribute assigned to a field. In those cases, if you want the value supplied to the field to be subject to directory-level validation, configure the field with an attribute that is mapped in directory.xml to the appropriate rule set.

How to Initiate Validation

At run time, validation is initiated in any of the following ways:

User submits a task

Validates the fields on the submitted task screen that are associated with validation rules.

User navigates to a different task screen tab

Validates the fields in the vacated tab that are associated with validation rules.

User clicks a Validate button on a tab

Validates the fields in the current tab that are associated with validation rules.

The Validate button also executes Logical Attribute Handlers that include the validate method.

User changes a value in a field who's Validate on Change property is yes

Validates the fields in the current tab that are associated with validation rules.

For example, if Validate on change is enabled for an Employee Type field, and the field value is changed from Non-exempt to Exempt, all fields on the tab that are associated with validation rules are validated. One rule could require that a Salary field contain a value, and another rule could automatically change an Hourly Rate field to 0.

Custom code uses a setAttribute... method in AttributeCollection or a tab handler to set a managed object attribute value

The field is configured with the managed object attribute being set.

Sample Implementations

Sample JavaScript implementations of validation rules are located in the following samples directory of your CA Identity Manager installation:

```
Identity Manager\samples\validationscripts
```

Chapter 7: Workflow API

This section contains the following topics:

Workflow API Overview (see page 121)
Participant Resolver API (see page 126)

Workflow API Overview

The Workflow API (../javadoc-im/com/netegrity/imapi/WorkflowContext.html) provides the following kinds of information to a custom script in a workflow process:

- Information about the currently executing CA Identity Manager event
- Information about the administrator who submitted the event
- Information within the CA 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 that is accessed from a workflow script

Note: For more information about using WorkPoint workflow, see the *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 CA Identity Manager event mapped to the workflow process, including CA 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 performs is determined by the outcome of the condition scripts, as follows:

- In the transition CA.COM SUFFIX, the condition script checks whether the user's email address ends in ca.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 ca.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 a CA Identity Manager object during a workflow process

In the previous use case example, suppose the condition script for the transition CA.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 ca.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 CA 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:

WorkflowContext (../javadoc-im/com/netegrity/imapi/WorkflowContext.html)

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 CA Identity Manager services, including managed object retrieval from the data store.

Extends EventContext.

EventContext (javadoc-im/com/netegrity/imapi/EventContext.html)

Provides read/write access to the managed objects in CA Identity Manager events, and lets you generate secondary events.

This interface is part of the core API functional model.

Extends IMPersistentContext.

IMPersistentContext (javadoc-im/com/netegrity/imapi/IMPersistentContext.html)

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.

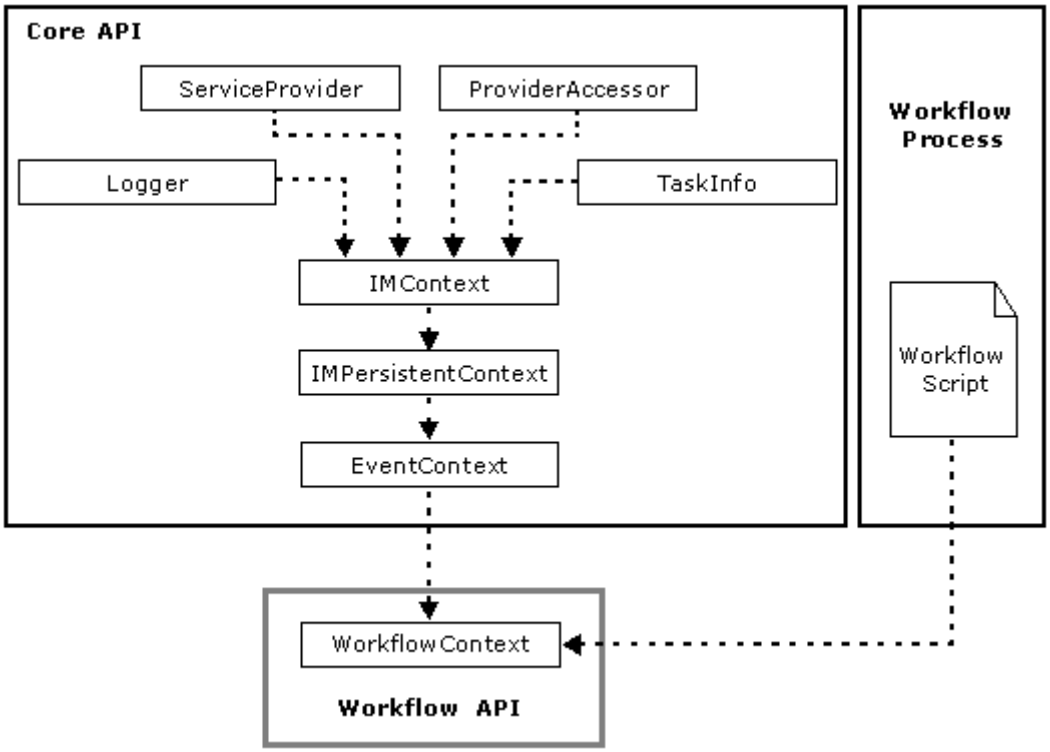
Note: There is no adapter component in the Workflow API because you do not create a custom Java object with this API. You call the methods in the API from a WorkPoint workflow script.

More Information:

Access to Event Objects (see page 153)

Workflow API Model

The components of the Workflow API are part of the CA Identity Manager API functional model, as shown in the following diagram:



Sample Workflow API Implementation

Sample Java classes that calls the Workflow API is in the samples\WorkflowContext subdirectory of your CA Identity Manager installation:

admin_tools\samples\WorkflowContext

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

Integration with Third-Party Applications

The Workflow API lets you integrate third-party applications with information in CA 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 CA Identity Manager objects to the application.

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

  In addition, the new user must 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.

  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 CA 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, 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 if 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 CA Identity Manager User object with the email address.

  When CA Identity Manager completes the execution of the CreateUserEvent event, the email address is stored with the other information about the new user.

Saving Job Objects

To prevent the possibility of a workflow job being saved to cache in a corrupt state when more than one operation occurs on the job simultaneously, the job object is cloned after being read from the cache so that each individual thread has its own copy of the job it is modifying.

This requires custom scripts to use the following workflow API objects:

Synchronous scripts

All synchronous scripts must use the *This<object>Data* when fetching userdata from an activity. The *This<object>Data* objects that are passed in have access to the live job, and any updates to the job made in the script are persisted after the transaction completes.

Synchronous scripts include resource scripts, synchronous agent scripts, and state rules.

Asynchronous scripts

For all asynchronous scripts executed by the general monitor, read-only versions of *This<object>Data* objects are passed in. Asynchronous scripts do not have access to the live job.

Asynchronous scripts include asynchronous agent scripts, mail scripts, and alert scripts.

Asynchronous agent scripts can use the *This<object>Data* to retrieve the real job object, make changes, and save the job in the script itself.

The following code example illustrates the required code changes for an asynchronous script.

Current code:

```
ThisJobData.setUserData("Test","Pass");
```

Revised code:

```
var jobTableId = ThisJobData.getJobID();
var job = Job.queryByID(ClientContext,jobTableId,true);
job.setUserData("Test","Pass");
job.save();
```

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 CA 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

CA 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

Participant Resolver Overview

A workflow activity must reference the list of participants (or approvers) 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*.

The following are the different 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 more information about other types of participant resolvers, see the *Administration Guide*.

Custom Participant Resolvers

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

Typically, you 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 Management Console.

More Information:

Participant Resolver Configuration (see page 131)

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 CA Identity Manager.

CA Identity Manager passes ParticipantResolverContext information into the base class. CA 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 following:

- 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 CA Identity Manager services, including managed object retrieval from the data store.

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

Extends EventROContext.

EventROContext

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

Extends IMContext.

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

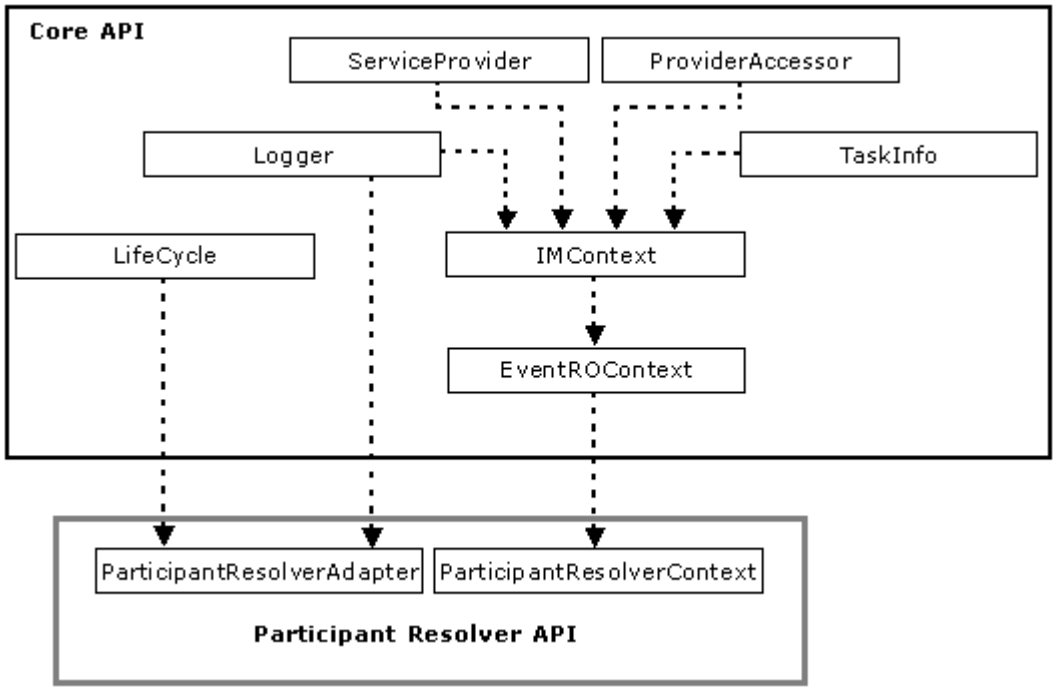
Note: For more information about these API components, see the *Javadoc Reference* (/javadoc-im/index.html).

More Information:

Access to Event Objects (see page 153)

Participant Resolver API Model

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



Sample Participant Resolver

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

admin_tools\samples\ParticipantResolver

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 CA Identity Manager. This information includes the type of participant resolver to use (as follows, by order of precedence), the name of the participant resolver (with custom types), and optional user-defined data.CA 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, CA Identity Manager selects one based on the following order of precedence:

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

CA 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 a CA Identity Manager-supplied script named IM Approvers. This script passes CA Identity Manager all the relevant property information defined within the workflow engine's user interface, and CA 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 a CA Identity Manager task.

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

Participant Resolver Configuration

Define a participant resolver's fully qualified class name and other properties in the Management Console.

To configuring a participant resolver:

1. In the Management Console, click 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 CA 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 173)

Chapter 8: Notification Rule API

This section contains the following topics:

API Overview (see page 133)
API Components (see page 134)
Email Templates (see page 136)
Notification Rule Configuration (see page 136)

API Overview

The Notification Rule API (javadoc-im/com/netegrity/imapi/NotificationRuleAdapter.html) lets you determine the users to receive an email notification regarding a CA Identity Manager event or task. An email notification's text and recipient list are generated from an email template.

CA Identity Manager includes the following 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

CA 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

CA Identity Manager objects

Object update

Not relevant

Can validate?

Not relevant

Use Case Examples

The following are 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.

API Components

The Notification Rule API contains the following components:

NotificationRuleAdapter
(javadoc-im/com/netegrity/imapi/NotificationRuleAdapter.html)

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
(javadoc-im/com/netegrity/imapi/NotificationRuleContext.html)

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 CA 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 (javadoc-im/com/netegrity/imapi/EventROContext.html)

Provides read/write access to the managed objects in CA Identity Manager events.

This interface is part of the core API functional model.

Extends IMContext.

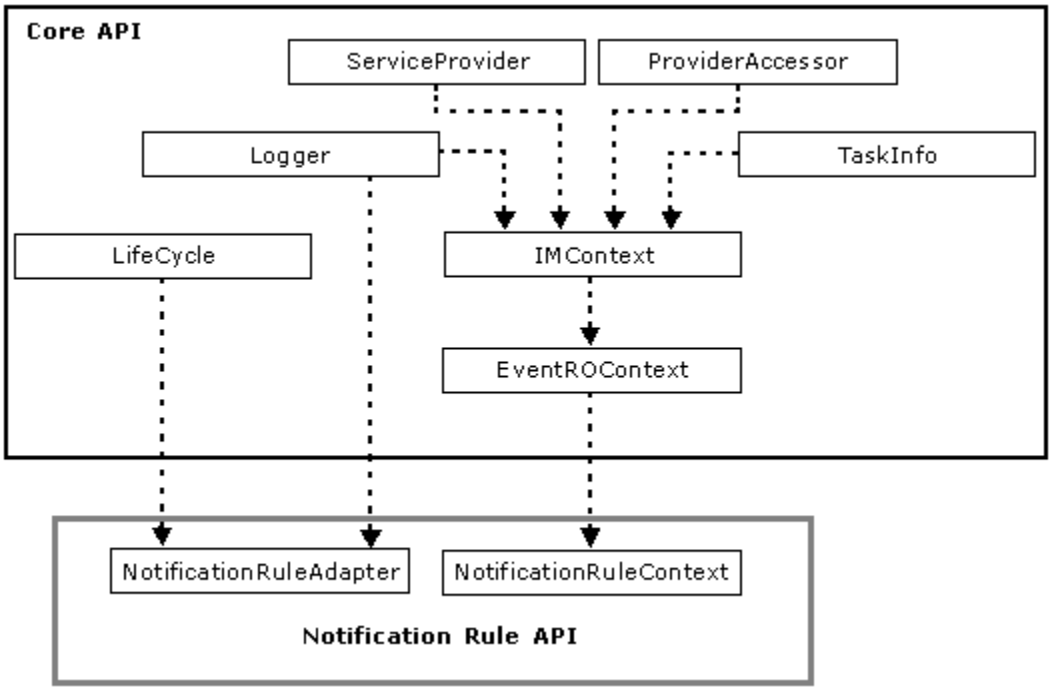
The previous components are in the package com.netegrity.imapi (javadoc-im/com/netegrity/imapi/package-summary.html).

More Information:

Access to Event Objects (see page 153)

Notification Rule API Model

The components of the Notification Rule API are part of the CA Identity Manager API functional model, as shown in the following diagram:



Sample Notification Rule

A sample implementation of a notification rule is in the samples\NotificationRuleAdapter subdirectory of your CA Identity Manager installation:

admin_tools\samples\NotificationRuleAdapter

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

Email Templates

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

CA 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 (javadoc-im/com/netegrity/imapi/ExposedEventContextInformation.html)
- ExposedTaskContextInformation (javadoc-im/com/netegrity/imapi/ExposedTaskContextInformation.html)

These objects can be accessed directly from an email template through JavaScript. You do not have to create and compile custom Java objects.

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

Notification Rule Configuration

Define a notification rule's fully qualified class name and other properties in the Management Console.

To navigate to the Notification Rules screen

1. In the Management Console, click 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 CA Identity Manager.
- Modify an existing notification rule.
- Delete a notification rule.

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

Chapter 9: Managed Objects

This section contains the following topics:

Managed Object Overview (see page 139)
Managed Object Interfaces (see page 140)
Access to Managed Object Data (see page 144)
Access to Objects in the Data Store (see page 146)
Access to Objects in a Task (see page 150)
Access to Event Objects (see page 153)

Managed Object Overview

CA Identity Manager administrators perform admin tasks against managed objects. The CA 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 CA Identity Manager Directory for a particular CA Identity Manager environment (User, Group, Organization).

  A CA 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 CA Identity Manager object-class is defined.

- Proprietary objects defined within a CA Identity Manager environment (Admin Role, Admin Task, Access Role, Access Task, Password Policy).

- Objects defined in the Provisioning Directory associated with the CA 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 CA Identity Manager managed objects are accessible through CA Identity Manager APIs:

AccessRole
(javadoc-im/com/netegrity/llsdk6/imsapi/managedobject/AccessRole.html)

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

AccessTask
(javdoc-im/com/netegrity/llsdk6/imsapi/managedobject/AccessTask.html)

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

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

AdminRole
(javadoc-im/com/netegrity/llsdk6/imsapi/managedobject/AdminRole.html)

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

AdminTask
(javadoc-im/com/netegrity/llsdk6/imsapi/managedobject/AdminTask.html)

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

Group (javadoc-im/com/netegrity/llsdk6/imsapi/managedobject/Group.html)

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

Organization
(javadoc-im/com/netegrity/llsdk6/imsapi/managedobject/Organization.html)

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
(javadoc-im/com/netegrity/llsdk6/imsapi/managedobject/PasswordPolicy.html)

Represents a set of restrictions and controls around passwords.

ProvisioningRole (javadoc-im/com/netegrity/llsdk6/imsapi/managedobject/ProvisioningRole.html)

Represents the definition of a provisioning role to CA Identity Manager.

This object requires an instance of a provisioning directory.

Role (javadoc-im/com/netegrity/llsdk6/imsapi/managedobject/Role.html)

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, similar to the 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 there are any attribute changes to commit.

Task (javadoc-im/com/netegrity/llsdk6/imsapi/managedobject/Task.html)

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

User (javadoc-im/com/netegrity/llsdk6/imsapi/managedobject/User.html)

Represents an individual within an organization in a user directory.

All interfaces are in this package:

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

More Information:

Super Interfaces (see page 142)

Super Interfaces

All CA Identity Manager managed objects have the following super interfaces:

ManagedObject
(javadoc-im/com/netegrity/llsdk6/imsapi/managedobject/ManagedObject.html)

Lets you access a managed object that extends this interface.

Package: com.netegrity.llsdk6.imsapi.managedobject

ModifiableObject
(javadoc-im/com/netegrity/llsdk6/imsapi/abstractinterface/ModifiableObject.html)

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
(javadoc-im/com/netegrity/llsdk6/imsapi/abstractinterface/AttributeCollection.html)

Provides access to managed object attributes as name/value pairs. The attributes you can access correspond to the fields and values on a CA 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
(javadoc-im/com/netegrity/llsdk6/imsapi/abstractinterface/NamedObject.html)

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 roles and tasks.

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

CA Identity Manager managed objects extend ManagedObject (javadoc-im/com/netegrity/llsdk6/imsapi/managedobject/ManagedObject.html), and ManagedObject extends AttributeCollection (javadoc-im/com/netegrity/llsdk6/imsapi/abstractinterface/AttributeCollection.html).

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 154)

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.

Example:

```
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 CA Identity Manager APIs, for example:

- To the Business Logic Task Handler API, when validating user-supplied data on a task screen against attributes of CA Identity Manager objects
- To the Workflow API, when using attributes of CA 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 CA Identity Manager objects during the following event states: Pre-approval, Approved, and Rejected
  - Retrieving attributes of CA Identity Manager objects during the Post-execution event state

You can access managed objects in the following ways:

ProviderAccessor

Interface

ProviderAccessor (javadoc-im/com/netegrity/imapi/ProviderAccessor.html)

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 (javadoc-im/com/netegrity/imapi/BLTHContext.html)

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.

CA 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 (javadoc-im/com/netegrity/imapi/EventContext.html)

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.

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

EventROCContext

Interface

EventROContext (javadoc-im/com/netegrity/imapi/EventROContext.html)

API

Notification Rule Participant Resolver

Comments

Read/write access.

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

CA 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 Event Objects (see page 153)
Access to Objects in the Data Store (see page 146)
Access to Objects in a Task (see page 150)

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 must 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 CA 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 CA 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
```

CA Identity Manager includes the following providers:

- AccessControlProvider (javadoc-im/com/netegrity/llsdk6/imsapi/provider/AccessControlProvider.html)
- AccessRoleProvider (javadoc-im/com/netegrity/llsdk6/imsapi/provider/AccessRoleProvider.html)
- AccessTaskProvider (javadoc-im/com/netegrity/llsdk6/imsapi/provider/AccessTaskProvider.html)
- AdminRoleProvider (javadoc-im/com/netegrity/llsdk6/imsapi/provider/AdminRoleProvider.html)
- AdminTaskProvider (javadoc-im/com/netegrity/llsdk6/imsapi/provider/AdminTaskProvider.html)
- GroupProvider (javadoc-im/com/netegrity/llsdk6/imsapi/provider/GroupProvider.html)
- OrganizationProvider (javadoc-im/com/netegrity/llsdk6/imsapi/provider/OrganizationProvider.html)
- ProvisioningPolicyProvider (javadoc-im/com/netegrity/llsdk6/imsapi/provider/ProvisioningPolicyProvider.html)
- ProvisioningRoleProvider (javadoc-im/com/netegrity/llsdk6/imsapi/provider/ProvisioningRoleProvider.html)

- SecurityProvider (javadoc-im/com/netegrity/llsdk6/imsapi/provider/SecurityProvider.html)
- SynchronizationProvider (javadoc-im/com/netegrity/llsdk6/imsapi/provider/SynchronizationProvider.html)
- UserProvider (javadoc-im/com/netegrity/llsdk6/imsapi/provider/UserProvider.html)

Retrieving Providers

ProviderAccessor (javadoc-im/com/netegrity/imapi/ProviderAccessor.html) is one of the core objects in the CA 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 CA 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 CA Identity Manager events are generated because the managed object is not part of a task session. Managed object updates are reflected in CA 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 150)

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. (javadoc-im/com/netegrity/llsdk6/imsapi/collections/AttributeRightsCollection.html)

AttributeRightsCollection is a collection of AttributeRight (javadoc-im/com/netegrity/llsdk6/imsapi/metadata/AttributeRight.html) objects.

Permission Request Rules

CA 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 applies 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 as follows:
   - 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 143)

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 can 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 telephone 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 (javadoc-im/com/netegrity/imapi/BLTHContext.html), 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, CA 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, CA 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 CA 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 CA 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 (javadoc-im/com/netegrity/imapi/BLTHContext.html) get... method and then update the object, the update is represented by one or more events that CA 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 153)

Calling Sequence

To access and update a managed object in a task session

1. Call one of the get... methods in BLTHContext (javadoc-im/com/netegrity/imapi/BLTHContext.html) to retrieve the managed object to be updated.
2. Manage the task's subject and its relationships through AttributeCollection (javadoc-im/com/netegrity/llsdk6/imsapi/abstractinterface/AttributeCollection.html).

More Information:

Managed Object Attributes (see page 143)

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, CA Identity Manager applies the rule when the attribute value is set.

CA Identity Manager enforces validation rules when a business logic task handler calls a setAttribute... method in AttributeCollection to set an attribute. However, CA Identity Manager only enforces directory-level validation for attributes set in this way. The reason is that 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 170)

Access to Event Objects

When a CA 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), CA Identity Manager breaks down the task into its component events.

These events can be accessed by custom objects that CA 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 a CA Identity Manager event. EventContext also allows the generation of secondary events:

EventContext (javadoc-im/com/netegrity/imapi/EventContext.html)

- 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 (javadoc-im/com/netegrity/imapi/EventROContext.html)

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

If you modify a managed object in an event, CA 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 CA Identity Manager event is represented by an event object. An event object has the same name as the corresponding event.

CA Identity Manager generates an object for each event and makes the objects available for retrieval through EventContext and EventROContext. When 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 (javadoc-im/com/netegrity/imapi/IMEventName.html) 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());
}
```

Base Event Interfaces

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

Base Interface	Method	Managed Object Retrieved
AccessRoleEvent (javadoc-im/com/netegrity/imapi/AccessRoleEvent.html)	getAccessRole()	AccessRole
AccessTaskEvent (javadoc-im/com/netegrity/imapi/AccessTaskEvent.html)	getAccessTask()	AccessTask
AdminRoleEvent (javadoc-im/com/netegrity/imapi/AdminRoleEvent.html)	getAdminRole()	AdminRole
AdminTaskEvent (javadoc-im/com/netegrity/imapi/AdminTaskEvent.html)	getAdminTask()	AdminTask

Base Interface	Method	Managed Object Retrieved
GroupEvent (javadoc-im/com/netegrity/imapi/GroupEvent.html)	getGroup()	Group
OrganizationEvent (javadoc-im/com/netegrity/imapi/OrganizationEvent.html)	getOrganization()	Organization
ParentOrganizationEvent (javadoc-im/com/netegrity/imapi/ParentOrganizationEvent.html)	getParentOrganization()	Organization
ProvisioningRoleEvent (javadoc-im/com/netegrity/imapi/ProvisioningRoleEvent.html)	getProvisioningRole()	ProvisioningRole
UserEvent (javadoc-im/com/netegrity/imapi/UserEvent.html)	getUser()	User

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

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.

When 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.

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 a CA Identity Manager environment. You can enable auditing for some or all the CA Identity Manager events generated by admin tasks.

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

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

admin_tools\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, 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 *Configuration Guide*.

Event Object Tables

To help you determine the get... methods and the managed objects contained in the current event object, you can use the Management Console.

To view event objects

1. In the Management Console, click Environments.
2. Select the appropriate environment.

   The Environment Properties page appears.
3. Click Advanced Settings.
4. Click Event List.

   All event objects in the environment are listed, with the implementing class, primary object type, and secondary object type.

More Information:

Base Event Interfaces (see page 154)

Chapter 10: Support Objects

This section contains the following topics:

Identity Policies (see page 159)
IMEvent (see page 160)
IMEventName (see page 160)
IMSException (see page 161)
Localizer Class (see page 163)
PasswordCondition (see page 163)
Permissions Objects (see page 164)
Policies and Policy Conditions (see page 164)
PropertyDict (see page 165)
Provisioning Roles (see page 165)
ResultsContainer (see page 166)
RoleDisplayData (see page 166)
Search Objects (see page 167)
Task.FieldId (see page 168)
TSContext (see page 168)
Type Objects (see page 168)
Validation Objects (see page 170)

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, CA Identity Manager can 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 polices that have already been allocated, and determining which policies must be allocated or deallocated by comparing the two lists.

The following objects manage synchronizing users with identity policies:

- AutoSynchType (javadoc-im/com/netegrity/llsdk6/imsapi/type/AutoSynchType.html)
- IdentityPolicySetEvent (javadoc-im/com/netegrity/imapi/IdentityPolicySetEvent.html)
- SynchronizationProvider (javadoc-im/com/netegrity/llsdk6/imsapi/provider/SynchronizationProvider.html)

Note: Some code element names can be based on the obsolete term "business policies" instead of "identity policies."

IMEvent

IMEvent (javadoc-im/com/netegrity/imapi/IMEvent.html) 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 153)

IMEventName

The IMEventName (javadoc-im/com/netegrity/imapi/IMEventName.html) interface contains constants for the names of CA Identity Manager events (typically, the constant name is the same as the corresponding event name, but in uppercase letters).

Package

com.netegrity.imapi

IMSException

The IMSException (javadoc-im/com/netegrity/ims/exception/IMSException.html) class contains one or more exception messages 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. The user can then 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 IMSException (javadoc-im/com/netegrity/ims/exception/IMSException.html) object are stored in resource bundles. Each resource bundle with the same base name contains the same messages, but in a different language.

CA Identity Manager includes a family of resource bundles containing CA Identity Manager system messages. These resource bundles have the base name IMSExceptions.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 do 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 IMSExceptions.properties. If you store custom messages in the IMSExceptions.properties file, do so at the end of the file, and in the format indicated in the file. For example, you can 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 IMSExceptions.properties file helps to avoid migration problems when you update CA Identity Manager.

IMSException (javadoc-im/com/netegrity/ims/exception/IMSException.html) messages support variables. You can insert specific values into the variables when you add the message to the IMSException 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 (javadoc-im/com/netegrity/imapi/TaskInfo.html) 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 IMSException constructor.
- To load a message into IMSException, call addUserMessage() in the IMSException (javadoc-im/com/netegrity/ims/exception/IMSException.html) 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, as follows:

- 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 can be named:

  LocalizedMessages_de.properties

  CA 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

We recommend that you store your custom resource bundles in the following location within the CA 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 *Configuration Guide*.

Localizer Class

Localizer (javadoc-im/com/netegrity/ims/util/Localizer.html) 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 *Configuration Guide*.

More Information:

IMSException (see page 161)

Use the Localizer Class

You can use the Localizer (javadoc-im/com/netegrity/ims/util/Localizer.html) object from your custom code or from a JSP file for a new skin.

To localize a new skin

1. Add the localized strings to the IMSResources bundles provided with CA Identity Manager and to any custom bundles you 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 (javadoc-im/com/netegrity/llsdk6/imsapi/utility/PasswordCondition.html) 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 listed following.

AttributeRight

The AttributeRight (javadoc-im/com/netegrity/llsdk6/imsapi/metadata/AttributeRight.html) class contains the name of an attribute and its associated permission.

Package

com.netegrity.llsdk6.imsapi.metadata

AttributeRightCollection

The AttributeRightCollection (javadoc-im/com/netegrity/llsdk6/imsapi/collections/AttributeRightsCollection.html) 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 168)

Policies and Policy Conditions

The following classes and interfaces let you manage admin policies, membership policies, and policy conditions:

- AdminPolicy (javadoc-im/com/netegrity/llsdk6/imsapi/policy/AdminPolicy.html)
- MemberRule (javadoc-im/com/netegrity/llsdk6/imsapi/policy/rule/MemberRule.html)
- MembershipPolicy (javadoc-im/com/netegrity/llsdk6/imsapi/policy/MembershipPolicy.html)
- Policy (javadoc-im/com/netegrity/llsdk6/imsapi/policy/Policy.html)

- ScopeRule (javadoc-im/com/netegrity/llsdk6/imsapi/policy/rule/ScopeRule.html)
- TriggerRule (javadoc-im/com/netegrity/llsdk6/imsapi/policy/rule/TriggerRule.html)

Note: Some code element names can 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 (javadoc-im/com/netegrity/llsdk6/imsapi/collections/PropertyDict.html) interface contains a tab's configuration parameters as defined in the User Console.

Package

com.netegrity.llsdk6.imsapi.collections

Provisioning Roles

The following objects let you define Provisioning roles, and to associate provisioning roles with Provisioning policies within CA Identity Manager:

- Account (javadoc-im/com/netegrity/llsdk6/imsapi/managedobject/Account.html)
- AccountHolder (javadoc-im/com/netegrity/llsdk6/imsapi/abstractinterface/AccountHolder.html)
- AccountReason (jaadoc-im/com/netegrity/llsdk6/imsapi/utility/AccountReason.html)
- ProvisioningPolicy (javadoc-im/com/netegrity/llsdk6/imsapi/policy/ProvisioningPolicy.html)
- ProvisioningPolicyProvider (javadoc-im/com/netegrity/llsdk6/imsapi/provider/ProvisioningPolicyProvider.html)

- ProvisioningRole (javadoc-im/com/netegrity/llsdk6/imsapi/managedobject/ProvisioningRole.html)
- ProvisioningRoleProvider (javadoc-im/com/netegrity/llsdk6/imsapi/provider/ProvisioningRoleProvider.html)

Note: Use of these objects requires an instance of a Provisioning Directory.

Package

Various.

ResultsContainer

The ResultsContainer (javadoc-im/com/netegrity/imapi/ResultsContainer.html) 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 can 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 (javadoc-im/com/netegrity/llsdk6/imsapi/metadata/RoleDisplayData.html) interface represents the information CA Identity Manager requires to display information about a role.

Package

com.netegrity.llsdk6.imsapi.metadata

Search Objects

Use the classes listed following to define and manage search operations in a CA Identity Manager environment.

- ApiResultSet (javadoc-im/com/netegrity/llsdk6/imsapi/search/ApiResultSet.html)
- RoleObjectQuery (javadoc-im/com/netegrity/llsdk6/imsapi/search/RoleObjectQuery.html)
- SearchCursor (javadoc-im/com/netegrity/llsdk6/imsapi/search/SearchCursor.html)
- SearchExpression (javadoc-im/com/netegrity/llsdk6/imsapi/search/SearchExpression.html)
- TaskObjectQuery (javadoc-im/com/netegrity/llsdk6/imsapi/search/TaskObjectQuery.html)

Package

com.netegrity.llsdk6.imsapi.search

Search Filters and Constraints

Use the classes listed following and interfaces to define search filters, membership constraints, and scope constraints.

- AttributeExpression (javadoc-im/com/netegrity/llsdk6/imsapi/policy/rule/constraints/AttributeExpression.html)
- GroupFilter (javadoc-im/com/netegrity/llsdk6/imsapi/policy/rule/constraints/GroupFilter.html)
- MemberMatchConstraint (javadoc-im/com/netegrity/llsdk6/imsapi/policy/rule/constraints/MemberMatchConstraint.html)
- OrgFilter (javadoc-im/com/netegrity/llsdk6/imsapi/policy/rule/constraints/OrgFilter.html)
- OrgMembershipConstraint (javadoc-im/com/netegrity/llsdk6/imsapi/policy/rule/constraints/OrgMembershipConstraint.html)
- OrgMembershipIdentifier (javadoc-im/com/netegrity/llsdk6/imsapi/policy/rule/constraints/OrgMembershipIdentifier.html)
- OrgScopeConstraint (javadoc-im/com/netegrity/llsdk6/imsapi/policy/rule/constraints/OrgScopeConstraint.html)

- OrgScopeIdentifier (javadoc-im/com/netegrity/llsdk6/imsapi/policy/rule/constraints/OrgScopeIdentifier.html)
- RuleConstraint (javadoc-im/com/netegrity/llsdk6/imsapi/policy/rule/constraints/RuleConstraint.html)
- UserFilter (javadoc-im/com/netegrity/llsdk6/imsapi/policy/rule/constraints/UserFilter.html)

Package

com.netegrity.llsdk6.imsapi.policy.rule.constraints

Task.FieldId

The Task.FieldId (javadoc-im/com/netegrity/llsdk6/imsapi/managedobject/Task.FieldId.html) 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 (javadoc-im/com/netegrity/llsdk6/imsapi/utility/TSContext.html) 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 (javadoc-im/com/netegrity/llsdk6/imsapi/type/AccountType.html)

Contains constants for provisioning account objects.

ActionType (javadoc-im/com/netegrity/llsdk6/imsapi/type/ActionType.html)

Contains constants that represent the type of operation being performed (for example, modify, delete, or approval operations).

AttributeChangeType (javadoc-im/com/netegrity/llsdk6/imsapi/type/AttributeChangeType.html)

Contains constants that represent the type of changes that can 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 (javadoc-im/com/netegrity/llsdk6/imsapi/type/ConjunctionType.html)

Contains constants used in search expressions.

DisabledReasonType (javadoc-im/com/netegrity/llsdk6/imsapi/type/DisabledReasonType.html)

Contains constants that indicate why a user account was disabled.

ObjectType (javadoc-im/com/netegrity/llsdk6/imsapi/type/ObjectType.html)

Contains constants that specify particular types of managed objects (for example, User, Organization, Group objects).

OperatorType (javadoc-im/com/netegrity/llsdk6/imsapi/type/OperatorType.html)

Contains constants that specify the operators that can be used in search expressions.

PasswordMessageType (javadoc-im/com/netegrity/llsdk6/imsapi/type/PasswordMessageType.html)

Contains constants that specify the reason a password is invalid.

PermissionType (javadoc-im/com/netegrity/llsdk6/imsapi/type/PermissionType.html)

Contains constants that represent the permissions assigned to directory attributes.

SearchDepthType (javadoc-im/com/netegrity/llsdk6/imsapi/type/SearchDepthType.html)

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, CA 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 the following ways:

- By directly associating a validation rule with a field on a task screen.

  This type of attribute validation is named *task-level validation*. CA Identity Manager validates the field value against other attributes in the task, for example, validating an area code for a user's telephone 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 named *directory-level validation*. CA 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 telephone number matches the nnn-nnn-nnnn format used in the directory attribute.

Note: For more information, see the *Configuration Guide*.

Attribute Validation

The following objects are used during attribute validation:

AttributeValidator
(javadoc-im/com/netegrity/llsdk6/imsapi/metadata/AttributeValidator.html)

The class for performing directory-level validation of managed object attributes. This abstract class is extended by directory-level validation rules that must access user, group, and organization objects in the user directory.

Package

com.netegrity.llsdk6.imsapi.metadata

AttributeValidationException
(javadoc-im/com/netegrity/llsdk6/imsapi/exception/AttributeValidationException.html)

The exception class for attribute validation errors.

Package

com.netegrity.llsdk6.imsapi.exception

DirectoryProvidersCollection (javadoc-im/com/netegrity/llsdk6/imsapi/utility/DirectoryProvidersCollection.html)

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 (javadoc-im/com/netegrity/llsdk6/imsapi/metadata/IAttributeValidator.html)

The interface for performing directory-level validation of managed object attributes. This interface is implemented by directory-level validation rules. If you must validate an attribute against information in the user directory, extend the AttributeValidator abstract class.

Package

com.netegrity.llsdk6.imsapi.metadata

IAttributeValidator.StringRef (javadoc-im/com/netegrity/llsdk6/imsapi/metadata/IAttributeValidator.StringRef.html)

A class that lets you return an updated attribute value or error message to CA Identity Manager during directory-level attribute validation.

Package

com.netegrity.llsdk6.imsapi.metadata

TaskValidator (javadoc-im/com/netegrity/imapi/TaskValidator.html)

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 (javadoc-im/com/netegrity/imapi/TaskValidator.StringRef.html)

A class that lets you return an updated attribute value or error message to CA Identity Manager during task-level attribute validation.

Package

com.netegrity.imapi

Chapter 11: Compiling and Deploying

This section contains the following topics:

Compiling Source Files (see page 173)
Deploying Run Time Files (see page 174)

Compiling Source Files

You can write custom Java objects that provide the following functionality:

- Logical attribute handlers
- Business logic task handlers
- Participant resolvers used in workflow operations
- Event listeners
- Email notification rules

Note: The instructions on compiling do not necessarily apply to the Workflow API and to the Email Template API. You may not need to compile Java objects with these APIs because the methods in the Workflow API can be called from either workflow scripts or Java objects, and the methods in implicit message objects are called from email templates.

When you write new Java files or modify existing ones, compile your Java files against all the JAR files located in the following CA Identity Manager lib directory:

admin_tools\lib

The default Windows path is:

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

For Java object make file examples, see the sample code directory:

admin_tools\samples

Note: These files (with CA Identity Manager API samples and other files) are installed when you select the CA Identity Manager Administrative Tools option during installation.

You may not need to compile Java objects with the following APIs:

- Workflow API. These methods can be called from workflow scripts instead of Java objects.
- Email Template API. These methods are called from email templates.

Deploying Run Time Files

The following sections describe the locations where you place custom run-time files.

Note: The slash character is platform-specific. On Windows platforms, use the backslash (\). On UNIX platforms, use the forward slash (/). The backslash character is used for the following directory examples.

Java Class Files

CA Identity Manager looks for compiled .class files in the following root location within your application server:

```
iam_im.ear\custom
```

For example, if your custom object is com.mycompany, the following is a valid location for myBLTH.class on a WebLogic deployment:

```
weblogic_home\user_projects\mydomain\applications\iam_im.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:

```
iam_im.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:

```
iam_im.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:

```
iam_im.ear\custom\emailTemplates
```

Chapter 12: Task Execution Web Service

This section contains the following topics:

Web Services Overview (see page 175)
About TEWS (see page 176)
Remote Task Requests (see page 177)
Task Operation Types (see page 179)
Web Service Configuration (see page 181)
Client Application Development (see page 184)
SOAP Messages (see page 187)
Exception Messages (see page 187)
Authentication Using SiteMinder (see page 188)
Authentication Using WSS Username Token (see page 189)
Authorization (see page 189)

Web Services Overview

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 named 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 various 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 CA Identity Manager:

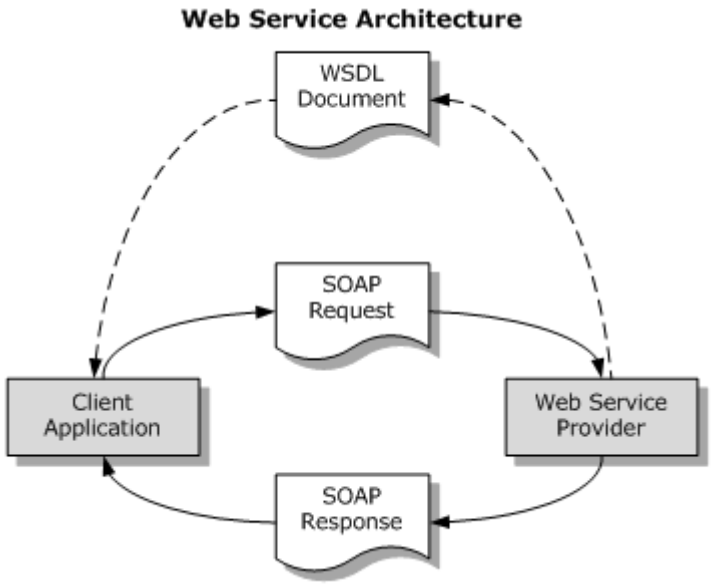
Simple Object Access Protocol (SOAP)

Defines the run-time 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:



About TEWS

The CA Identity Manager Task Execution Web Service (TEWS) is a web service interface that allows third-party client applications to submit remote tasks to CA Identity Manager for execution. This interface implements the open standards of WSDL and SOAP to provide remote access to CA Identity Manager. All CA 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 CA 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 CA Identity Manager. The WSDL document describes the metadata that the client application requires to prepare and submit a task request.

CA 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 CA Identity Manager to perform the web service task implementation. No CA Identity Manager APIs are provided for these purposes.

Remote Task Requests

CA 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 CA 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: CA 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 24)
Asynchronous Phase (see page 28)

How a Remote Request Is Processed

In this 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. With ViewUserQuery, the required information consists of the following:

   - The task tag
   - The CA 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 CA 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 CA Identity Manager tasks can be processed through the Task Execution Web Service.

Request a TEWS task from CA Identity Manager by invoking the following URL:

```
http://host:port/iam/im/TEWS6/environmentalias
```

For example:

```
http://tewstest.ca.com:7001/iam/im/TEWS6/neteauto
```

The URL is case-sensitive, and *environmentalias* is the protected alias of the CA Identity Manager environment.

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. CA Identity Manager 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 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 can 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 User Console.

Execute

Performs a CA 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 CA Identity Manager admin task using remote requests is a three-step process that requires the operation types search, query, and execute be called in the following sequence:

1. Retrieve a list of managed objects using the search operation which corresponds to the CA Identity Manager admin task.
2. Retrieve profile and relationship information for a particular managed object using the corresponding query operation.
3. Submit a remote request for CA Identity Manager to execute the admin action on the managed object.

Most admin task operations, such as Modify User, Modify Group, and Modify Admin Task, 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

Note: Not all CA Identity Manager admin tasks require the three-step process. The SelfRegistration task 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 CA 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 CA Identity Manager environment.
- In the User Console, the individual task must be enabled for web services.

Configure Web Service Properties

Configure web services properties for the CA Identity Manager environment in the Management Console.

To display the Web Services Properties screen

1. In the Management console, click Environments.
2. Click the name of the environment where the web services will be used.
3. Click Advanced Settings.
4. Click Web Services.

   The Web Service Properties page appears.

Web Services Properties Screen

The Web Services Properties screen lets you configure the Task Execution Web Service for the environment. This service allows third-party client applications to submit tasks from a remote location to CA Identity Manager for execution.

There are seven 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, restart the application server.

Enable WSDL Generation

Check to enable WSDL generation for the environment.

The URL to perform WSDL generation is accessible only when WSDL generation is enabled.

By default, WSDL generation is enabled.

Note: When generating WSDLs in a clustered environment, run one Policy Server and one CA Identity Manager Server. Stop all other Policy Servers and CA 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.

This option is ignored when using WSS authentication.

Admin password is required

Specifies that CA Identity Manager obtains the admin ID from the session information to determine whether the administrator is authorized to perform the task.

This field is ignored when using WSS authentication.

SiteMinder Authentication

When CA Identity Manager integrates 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, use an external method to secure the web services URL.

- **Basic Authentication**

SiteMinder basic authentication secures access to the web services URL.

When selected, CA Identity Manager automatically creates the authentication schemes, domain, realm, rule, and policies that are required 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 CA 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.

WSS UsernameToken (Password Text)

When selected, specifies that you are using the Web Services Security UsernameToken protocol for authentication. By default, this setting is not enabled.

Generate WSDL in WS-I form (Note: your existing TEWS code may need to be modified.)

When selected, specifies that the WSDL file is written compliant with the WS-I standard. By default, this setting is not enabled.

Note: None of the TEWS sample applications use this format.

When finished changing configuration settings on this screen, click Save.

For more information about issuing remote requests to CA Identity Manager through the Task Execution Web Service, see the *Programming Guide*. For more information about using SiteMinder to protect a web services URL, see the *CA SiteMinder Web Access Manager Policy Server Configuration Guide.*

Enable Web Services for Individual Tasks

In the User Console, verify that the tasks you want to invoke are enabled for web services.

To enable web services for tasks

1. Log into the 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. Select Enable Web Services.

   Note: This setting is selected by default.
7. Click Submit to save your changes.

Note: For more information about configuring tasks, see the *Administration Guide*.

Client Application Development

The following steps are an overview of the client application development process for submitting remote web service requests to CA Identity Manager.

Note: This overview assumes that your integrated development environment includes a third-party tool like Axis that generates proxies. While this approach automates some of the development, it is not required.

1. Be sure that web services are enabled for the CA Identity Manager environment and for each task that supports remote requests.

   Note: If you are using Axis 2, be sure to select the Generate WSDL in WS-I form option in Advanced Settings, Web Services, Web Services Properties Screen.
2. Generate the WSDL code by invoking the following URL:

   `http://host:port/iam/im/TEWS6/environmentalias?wsdl`

   For example:

   `http://tewstest.ca.com:7001/iam/im/TEWS6/neteauto?wsdl`

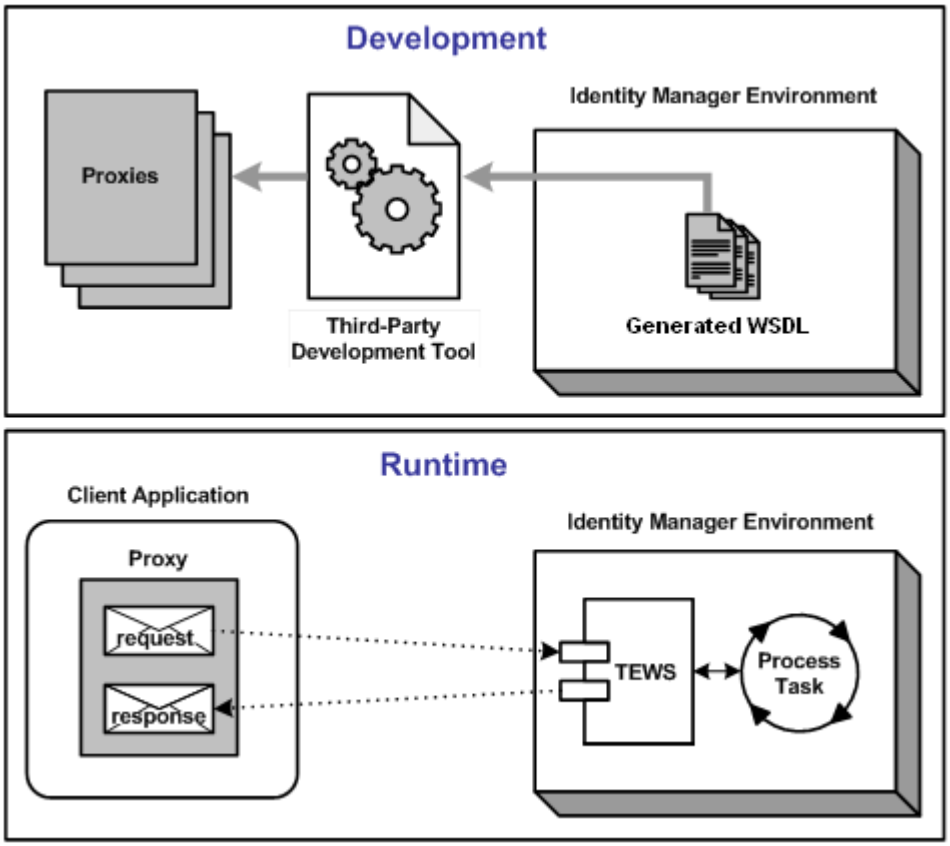
   Note: The URL is case-sensitive, and *environmentalias* is the protected alias of the CA Identity Manager environment.
3. In your integrated development environment (IDE), create the proxies based on the WSDL document generated in the preceding step.

   Proxies reflect the operations that your client can submit to CA 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 formulates and submits 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 required to process the SOAP requests and responses in compliance with your business requirements.

5. At run time, the client application submits a CA 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. CA 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

Request a WSDL document from CA Identity Manager by invoking the following URL:

`http://host:port/iam/im/TEWS6/environmentalias?wsdl`

For example:

`http://tewstest.ca.com:7001/iam/im/TEWS6/neteauto?wsdl`

The URL is case-sensitive, and *environmentalias* is the protected alias of the CA Identity Manager environment.

If WSDL generation is enabled for the CA Identity Manager environment, the response from CA 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 User Console. All tasks are disabled for web services by default.

An IDE like Eclipse Web Tools Platform (WTP) enables you to invoke the URL and display the WSDL file 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, run one Policy Server and one CA Identity Manager Server, and stop all other Policy Servers and CA Identity Manager Servers within the cluster.

More Information:

Configure Web Service Properties (see page 181)

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.

SOAP Limitations

SOAP requests use the pipe ("|") as a delimiter to separate values in multi-valued attributes. For example, for an address attribute you can have several values separated by a piper, such as 110 First Avenue | Boston | MA.

Do not use the pipe character as a delimiter for single-valued attributes. For example, a specification such as Name: John | Jones can cause the request to fail.

Exception Messages

If an error occurs during the processing of a remote request, CA Identity Manager reports the error to the client in standard SOAP fault format, as follows:

- 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, CA 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 197)

Authentication Using SiteMinder

The URL for generating a WSDL is typically protected by a SiteMinder Agent. If you need a more stringent Task Execution Web Service security model, consider using CA SOA Security Manager to secure it. SOA Security Manager is available separately from CA.

SiteMinder and SOA 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 SOA 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. CA Identity Manager can obtain the admin ID from this session information when it determines whether the administrator is authorized to perform the task.

More Information:

Configure Web Service Properties (see page 181)

Authentication Using WSS Username Token

As an alternative to authentication using SiteMinder, you can specify authentication using the Web Security Services (WSS) UsernameToken protocol. The WSS UsernameToken profile is an Oasis standard that specifies how a web services consumer can supply a token to identify itself to a provider. The token includes a user name optionally a password, or password equivalent.

You can enable this authentication protocol with the WSS UsernameToken (Password Text) option in the Web Services Properties dialog. You select this option (or not) the same way as you select the Admin password is required option.

UsernameToken security in TEWS is implemented using a Servlet Filter. IM uses a Servlet Filter (FrameworkAuthFilter) to perform UI authentication. The TewsSecurityFilter processes TEWS requests. IM calls the TewsSecurityFilter is for every TEWS request. The filter verifies the status of the environment web service configuration before inspecting the message. If UsernameToken security is not configured, the message is passed on untouched.

The TewsSecurityFilter is defined in user console web.xml file. If you want to disable the filter manually, comment out the following lines:

```
<filter>
    <filter-name>TEWSFilter</filter-name>
    <filter-class>con.netegrity.ims.tews6.servletfilter.TewsSecurityFilter</f
    ilter-class>
</filter>
<filter-mapping>
    <filter-name>TEWSFilter</filter-name>
    <url-pattern>/TEWS6/*</url-pattern>
</filter-mapping>
```

After you have commented out these lines, restart the application server. IM no longer verifies the status of the UsernameToken security configuration.

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 the following ways:

- As part of the session information that SiteMinder or SOA Security Manager inserts into the POST request header during the authentication response. This is the default.
- Through the < admin_id> element that is passed to CA 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 SOA Security Manager. 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, CA 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, CA 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:

- JBoss and WebLogic:
  `iam_im.ear\user_console_war\WEB-INF`
- WebSphere:
  `WebSphere-ear\iam_im.ear\user_console_war\WEB-INF`

Whether CA Identity Manager uses the ID specified in the session or specified 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

The following examples illustrate the results of the several possible combinations of settings for the administrator parameters.

If **use_admin_id = false**, and **require_sm_headers = true**, the admin ID is in the session. The ID has been authorized for this request by CA SiteMinder or CA SOA Security Manager.

The previous settings are the default settings.

With these settings, CA Identity Manager uses the ID in the session, even if 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>
```

If **use_admin_id = true**, and **require_sm_headers = false**, the admin ID is specified in < admin_id>. CA Identity Manager does not verify whether the administrator is authorized for the request.

With these settings, CA Identity Manager does not check the session information for an ID.

Typically, these settings are used when SiteMinder or SOA 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>
```

If **use_admin_id = true**, and **require_sm_headers = true**, the admin ID is specified in < admin_id>. CA Identity Manager does not verify whether the administrator is authorized for the request.

With these settings, CA 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

The two segments in the channel from the client application to the Task Execution Web Service are as follows:

- From the client to the Web Server
- From the J2EE application server Web Server proxy to the J2EE server

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.

Chapter 13: TEWS Sample Client Code

This section contains the following topics:

Web Service Development Workstation (see page 193)
Web Service Samples (see page 194)
Java Samples for Axis (see page 195)
Build Script for Axis (see page 196)
Running the Java Classes (see page 196)

Web Service Development Workstation

Your CA Identity Manager web service development workstation must include installations of the following third-party software:

- Apache Ant 1.5 (or later)
- Apache Axis 1.3 (or later)
- Apache Axis 2.1.5 (or later)
- Eclipse WTP (or other IDE)
- Sun J2SDK/J2RE 1.4.2_7

On the development system, create a Java project directory that contains the web service sample files installed with CA Identity Manager.

The development system does not need to be the same system where CA 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 must 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 2.1.5 (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 CA 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

CA Identity Manager ships with a number of samples for common use cases. The samples are located in the following installation directory:

admin_tools\samples\WebService

Note the following:

- The Axis subdirectory contains Java client samples that use proxies generated by Apache Axis.

  Note: If you are using Axis 2 samples, select the Generate WSDL in WS-I form option in Advanced Settings, Web Services, Web Services Properties Screen.

- The .NET subdirectory contains C# client samples that demonstrate how to invoke CA Identity Manager tasks using web services.
- The XML subdirectory contains samples of task requests and responses formatted as SOAP documents.

Readme.txt files that contain prerequisites and instructions for using the samples are also included.

Java Samples for Axis

The Java samples for Axis demonstrate how to invoke CA 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 CA Identity Manager WSDL file.

The following samples are a set of Java classes for performing basic administrative tasks:

- Creating a user
- Changing a password
- Modifying user data

A directory of optional sample classes that require additional configuration of the NeteAuto environment is also included. 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 CA 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% well-known 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:

admin_tools\samples\WebService\Axis\build.xml

The following file provides instructions about how to configure and run the script:

admin_tools\samples\WebService\Axis\Readme.txt

Running the Java Classes

The process for building and running the Java samples for Axis is as follows:

1. Verify that your system 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:

admin_tools\samples\WebService\Axis\Readme.txt

Appendix A: TEWS Error Codes

CA Identity Manager reports web service-specific exception information in the < ImsException> element of the SOAP response. The < ImsException> information is presented in the < Detail> element of a SOAP fault.

The following table lists the CA 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 CA 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 CA 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

The following table lists the task failure error codes returned by CA 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 CA Identity Manager environment.
503	IME not found	Unable to find a CA Identity Manager 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.

Appendix B: Custom Authentication Schemes

This section contains the following topics:

How To Customize CA Identity Manager Authentication (see page 199)

How To Customize CA Identity Manager Authentication

When not using CA CA SiteMinder®, CA Identity Manager provides its own user authentication. You can customize this authentication scheme by adhering to the following process:

1. Modify the JSP-based login page credential form to suit your authentication requirements.
2. Write a module in Java that extends the AuthenticationModule interface so that it implements your changes to the login page.
3. Configure the Java class name and the login page name through the Management Console.

Modify the Login Credential Form

By default, the CA Identity Manager authentication scheme accepts the user name and password at login, provided in a credential form in the login.jsp file. These parameters are tested against credentials in the directory configured for the protected environment.

You can modify login.jsp to suit your authentication requirements. A partial listing login.jsp is shown following.

```
.
<form NAME="Login" METHOD="POST" target="_top">
. .
User Name: <input type="text" name="username" />
Password:<input type="password" name="password" />
. .
 </form>
. .
```

To modify the credential form

1. Edit login.jsp (located in iam_im.ear\user_console.war) as your authentication requirements demand.
   For example, you can substitute social security number for user name as follows:

   ```
   Social Security Number: <input type="text" name="socsecnum" />
   ```

2. Save the changes to a different file name in case you want to return to the default authentication scheme at a later time. Remember that the default login.jsp gets overwritten when you apply an upgrade.

   The login page is ready for your environment.

Implement the AuthenticationModule Interface

Write a custom authentication module that extends com.netegrity.webapp.authentication.AuthenticationModule, as follows:

```
package com.netegrity.webapp.authentication;

/**
 * Implement this interface to write a pluggable authentication module for use with the Framework Native auth.
 * The implemented class typically goes hand in hand with a login.jsp/html page that collects some information.
 * This information is passed along to the AuthenticationModule for processing. Typical information captured can include
 * userid and password.
**/
```

```
public abstract class AuthenticationModule
{
        /**
         * The httpSession attribute name where the exception from the authenticate
method will be available.
         */
        public static final String FWAUTH_EXCEPTION = "IAMFW_LOGIN_EXCEPTION";
        public static Vector MANDATORY_USER_ATTRIBS = null;
        public static Log _log = null;

        static
        {
        _log = LogFactory.createLog("im.AuthenticationModule");

        MANDATORY_USER_ATTRIBS = new Vector();
        //mandatory attribs for a user object
        MANDATORY_USER_ATTRIBS.add(User.PROPERTY_ENABLED_STATE);
        MANDATORY_USER_ATTRIBS.add(User.PROPERTY_FRIENDLY_NAME);
        }

        public AuthenticationModule()
        {
        }
```

```
        /**
         * This method will be called first by the FrameworkLoginFilter. With the given
set of information
         * in the login.jsp/html, the AuthenticationModule should be able to find a
User in the given ImsDirectory.
         *
         * @param request - The request object
         * @param response - The response object
         * @param env - The environment being accessed.
         * @return The user as found in the provided ImsDirectory.
         * @throws Exception - This exception will be put in the httpSession
        * as an attribute by the name FWAUTH_EXCEPTION
         */
        public abstract User disambiguateUser(HttpServletRequest request,
HttpServletResponse response, ImsEnvironment env) throws Exception;

        /**
         * @param request - The request object
         * @param response - The response object
         * @param env - The environment being accessed.
         * @return The user as found in the provided ImsDirectory.
         * @throws Exception - This exception will be put in the httpSession
        * as an attribute by the name FWAUTH_EXCEPTION
         */
        public abstract boolean authenticate(HttpServletRequest request,
HttpServletResponse response, ImsEnvironment env, User user) throws
FwAuthenticationException;
}
```

The default authentication module is listed here for reference. You can write your own authentication module using the default as a model. In general, you must be able to find and return a valid user in the directory of the CA Identity Manager environment being protected using the form and header variables.

Note: A sample authentication module, which you can also use as a model, is available in the following location:

admin_tools/samples/AuthenticationModule

```
package com.netegrity.webapp.authentication;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import com.netegrity.llsdk6.imsapi.exception.FwAuthenticationException;
import com.netegrity.llsdk6.imsapi.exception.NoSuchObjectException;
import com.netegrity.llsdk6.imsapi.managedobject.User;

import com.netegrity.llsdk6.imsapi.ImsDirectory;
import com.netegrity.llsdk6.imsapi.ImsEnvironment;
import com.netegrity.sdk.apiutil.SmApiException;

/**
 * The default Framework Authentication module. This works in conjunction
 * to the default login.jsp page. The Attribute to be used for looking up
 * the user is %USER_ID%.
 *
 */
public class DefaultAuthenticationModule extends AuthenticationModule {
        public static final String FORM_VAR_USERNAME="username";
        public static final String FORM_VAR_PASSWORD="password";

        public User disambiguateUser(HttpServletRequest request,
HttpServletResponse response, ImsEnvironment env) throws Exception
        {
                String username = request.getParameter(FORM_VAR_USERNAME);

                User user = null;
                try
                {
                        ImsDirectory dir = env.getImsDirectory();
                        user = dir.getUserProvider().disambiguateUser(username,
MANDATORY_USER_ATTRIBS.elements());
                }
                catch (NoSuchObjectException nsoe)
                {
                        throw new FwAuthenticationException("Username and password
do not match.");
                }
```

```
                return user;
        }
        public boolean authenticate(HttpServletRequest request, HttpServletResponse
response, ImsEnvironment env, User user) throws FwAuthenticationException
        {
                String password=request.getParameter(FORM_VAR_PASSWORD);
                //verify the user against the directory.

                boolean authenticated= false;
                try
                {
                        authenticated = user.authenticate(password);
                }
                catch (SmApiException e)
                {
                        _log.logDebug("Exception while authenticating:
"+e.getMessage());
                        _log.logDebug(e);
                        throw new FwAuthenticationException(e.getMessage());
                }
                if (!authenticated)
                {
                        throw new FwAuthenticationException("Username and password
do not match.");
                }
                return authenticated;
        }
}
```

Save your compiled Java class file to the iam_im.ear\custom folder.

Configure the Java Class and Login Page

The login form file and the authentication module are specified for an environment using the Management Console.

To configure the authentication provider class and the login page

1. In the Management Console navigate to the Advanced Settings, User Console pane for the particular environment.
2. Enter the fully qualified Class name of the compiled Java module in the Framework Native Login Properties group box.

3. Enter the name of the login page in the same group box.
4. Click Save.

   The custom authentication scheme is configured.

Index

A

access
  events • 153
  managed objects • 144
access tasks
  in managed objects • 140
  reserved fields • 168
AccessRole interface • 140
AccessRoleEvent interface • 154
AccessTask interface • 140
AccessTaskEvent interface • 154
ActionType class • 168
adapter base class
  Business Logic Task Handler API • 75
  Logical Attribute API • 48
  Notification Rule API • 134
  Participant Resolver API • 129
adaptor objects • 36
addMessageObject() • 78
addUserMessage() • 162
ADMIN notification rule • 133
administrator
  eTrust SiteMinder • 190, 191
  eTrust TransactionMinder • 190, 191
  ID • 190, 191
  policy • 164
  roles • 140
  rules • 164
  tasks • 24, 139, 140
AdminRole interface • 140
AdminRoleEvent interface • 154
AdminTask interface • 140
AdminTaskEvent interface • 154
after() • 86, 88, 91
Apache Axis • 194, 196
API
  adaptor objects • 36
  asynchronous phase model • 35
  Business Logic Task Handler • 75
  context objects • 36
  Email Template • 136
  examples of use • 23, 32
  functional model components • 36
  functional model overview • 34
  in asynchronous operations • 29
  in synchronous operations • 25
  Logical Attribute • 48
  Notification Rule • 134
  overview • 21
  Participant Resolver • 129
  purpose • 139
  synchronous phase model • 34
  Workflow • 123
ApiResultSet class • 167
approval task • 129
approvals
  for generated events • 151
  workflow • 22
approved() • 86, 91
APPROVERS_REQUIRED • 129
AssignableObject interface • 142
assignResources() • 72
asynchronous phase • 24, 28, 29, 35
Attribute Name dropdown list • 39, 53
attribute validation
  about • 170
  business logic task handlers • 152
  directory-level • 170
  exceptions • 161
  JavaScript • 46, 69
  objects used in • 170
  regular expression • 46, 69
  task data • 72
  task-level • 170
  user input • 47, 50
AttributeChangeType class • 168
AttributeCollection interface • 142, 143, 149
AttributePermissionException • 149
AttributeRight class • 148, 164
AttributeRightsCollection class • 148, 164
attributes
  about • 39
  committing changes • 144
  expressions • 167
  including in a retrieved object • 148
  logical and physical • 40, 42, 143
  managed object • 22, 143, 144
  mapping, in directory.xml • 41
  permissions • 148, 149, 164

provider access • 148
See also logical attributes, physical attributes, w • 39
task context access • 152
AttributeValidationException class • 170
AttributeValidator class • 170
AuditEvent interface • 156
auditing
changes to be included • 168
events • 151, 156
unavailable with providers • 146
Axis client • 194, 196

B

base resource bundle • 162
event descriptions • 156
IMSException • 161
IMSResources • 156, 163
BASE search depth type • 168
before() • 86, 90
BLTHAdapter class • 75
BLTHContext interface • 70, 75
managed object access • 144
Business Logic Task Handler API
access to managed objects • 144
BLTH. See Business Logic Task Handler API, business logic task • 69
called by CA Identity Manager • 69
calling sequence • 77
compiling • 173
data persistence • 75
handleSetSubject() • 25, 32
handleStart() • 25, 32
handleSubmission() • 25, 32
handleValidation() • 25, 32
objects • 75
sample • 76
task example • 32
use cases • 72
business logic task handlers
alternative validation methods • 69
attribute validation • 152
configuring • 74
executed after logical attribute • 50, 77
global • 73
JavaScript implementation • 81
managed object access • 80
order of execution • 77
parallels with task screen operat • 150
regular expression validation • 69
sample • 76
scope • 73
sharing data with other objects • 75
task session • 70, 150
task-specific • 73
validation • 152
business policies. See identity policies • 159

C

CA Identity Manager APIs
Business Logic Task Handler API • 75
Email Template API • 136
Logical Attribute API • 48
Notification Rule API • 134
Participant Resolver API • 129
Workflow API • 123
CA Identity Manager Directory • 139
calling sequence
Business Logic Task Handler API • 77
Logical Attribute API • 50
Participant Resolver API • 130
calls to
Business Logic Task Handler API • 69
Logical Attribute API • 46, 50
Notification Rule API • 133
casting events • 154, 160
committing changes to attributes • 140, 144
compiling source files • 173
configuring
business logic task handlers • 74
forgotten password handler • 68
logical attributes • 52
notification rules • 136
option lists • 60, 61
participant resolvers • 131
Confirm Password Handler • 43
ConjunctionType class • 168
constraints for searches • 167
contacting technical support • 3
context objects • 36
Custom type participant resolver • 127
order of precedence • 130
customer support
customer support, contacting • 3
customizing CA Identity Manager
Business Logic Task Handler API • 32, 75

examples • 23, 32
localization • 163
Logical Attribute API • 32, 42, 48
Notification Rule API • 32, 134
Participant Resolver API • 32, 128, 129
Workflow API • 32, 123

D

data flow • 42
data persistence
Business Logic Task Handler API • 75
Logical Attribute API • 48
Workflow API • 123
data store • 39
direct access through providers • 146
when data is committed • 144, 151
decrypt() • 67
deploying files • 174
directory.xml • 41
DirectoryProvidersCollection interface • 170
DisabledReasonType class • 168
dotNET sample files • 194
Drop Down Menu • 60

E

EAR file • 174
email notifications • 136
Email Template API
about • 136
and compilation • 173
task example • 32
email templates
automatic generation of email • 136
deploying • 174
recipient list • 133
enable logical attribute • 43
Enable User Handler. • 43
%ENABLED_STATE% • 43
encrypt() • 67
EnhancedAuditData class • 156
eTrust Admin provisioning • 139, 140, 165
eTrust SiteMinder • 188, 189, 190, 191
eTrust TransactionMinder • 188, 189, 190, 191
Event Listener API
access to managed objects • 144
compiling • 173
components • 89
EventContext • 89
EventListenerAdaptor • 89
sample code • 93
summary • 86
task example • 32
use cases • 88
event listeners
and managed objects • 94
and tasks • 85
configuration • 95
event generation • 95
execution • 90, 93
invocation rules • 93
sample code • 93
states • 86, 91
EventContext interface • 89, 90
API model components • 36, 89, 90
asynchronous phase model • 35
generateEvent() • 94
managed object access • 94, 144
read/write access • 153
synchronous phase model • 34
Workflow API • 123
EventListenerAdaptor • 89, 90, 91
EventProfile class • 156
EventROContext interface
API model components • 36
asynchronous phase model • 35
managed object access • 144
Notification Rule API • 134
Participant Resolver API • 129
read/write access • 153
synchronous phase model • 34
events
access to • 153
accessing managed objects through • 153, 154
AddToGroupEvent • 25, 85, 95
AssignAccessRoleEvent • 85
AssignAdminRoleEvent • 88
auditing • 151, 156
base interfaces for • 154, 157
casting a particular event • 154, 160
categories • 157
CreateUserEvent • 25, 85, 88
generating • 123, 144, 153
in asynchronous operations • 29
in asynchronous phase model • 35
in synchronous operations • 25
in synchronous phase model • 34
in tasks • 85

localizing event descriptions • 156
ModifyUserEvent • 88, 95
names • 154, 160
not generated with providers • 146
objects • 153, 157
overview • 25
RemoveFromGroupEvent • 94
when generated • 151
exceptions
displaying • 79
IMSException class • 161
remote request • 187, 198
storing • 161
ExposedEventContextInformation • 136, 153
ExposedTaskContextInformation • 136
external tasks • 154
extractAuditData() • 156

F

Field properties box • 53
fields
option list • 60
reserved • 168
Filter type participant resolver • 127
order of precedence • 130
filters for searches • 167
Forced Password Reset Handler • 43
forcePasswordReset logical attribute • 43
Forgotten Password Handler • 43, 66, 68
customizing • 67
Forgotten Password task • 67

G

generateEvent() • 94
generating secondary events • 123, 144, 153
GenericAuditEvent class • 156
getAdminTask() • 151
getDisabledUsers() • 151
getEnabledUsers() • 151
getEvent() • 154
getEventName() • 154
getGroupMembers() • 151
getNotifiers() • 29, 133
getOrganization() • 64
getQuestions() • 67
getUser() • 151
getUserLocale() • 162
global business logic task handlers
about • 73
configuring • 74
order of execution • 77
Grantor interface • 142
group events • 157
Group interface • 140
GROUP object type • 43
Group type participant resolver • 127
order of precedence • 130
GroupableObject interface • 142
GroupEvent interface • 154
groups
constraints • 167
managed object • 140
membership • 142
subject's relationships • 150
tabs and • 150
GroupSubscription logical attribute • 43

H

handlers. See business logic task handlers, logical attribute h • 42
handleSetSubject JavaScript method • 82
handleSetSubject() • 32, 72, 77
handleStart JavaScript method • 82
handleStart() • 25, 32, 77
handleSubmission JavaScript method • 82
handleSubmission() • 25, 32, 77
handleValidation JavaScript method • 82
handleValidation() • 25, 32, 77

I

IAttributeValidator interface • 170
IAttributeValidator.StringRef class • 170
identifier for organizations • 128
identity policies • 159
IdentityMinder.ear • 174
IM Approvers • 130
IMContext interface • 34, 35, 36, 89, 90
IMEvent interface • 154, 160
IMEventName interface • 160
IMPersistentContext interface • 89
API model components • 36
asynchronous phase model • 35
Business Logic Task Handler API • 75
Event Listener API • 90
Logical Attribute API • 48
Logical Attribute Handler API • 48

synchrouous phase model • 34
Workflow API • 123
ims.jar • 173
imsapi6.jar • 173
IMSException class • 161, 198
IMSExceptions.properties • 161
ImsManagedObjectAttr element • 41
IMSResources bundles • 156, 163
IMSStatus • 178
initialize()
default logical attribute values • 50
initializeOptionList() • 50, 60
integration with third-party applications • 125
invalid passwords • 163, 168
ISO 639 language identifiers • 162

J

JAR files • 173
JavaScript
attribute validation • 46, 69
business logic task handlers • 25, 81
deploying files • 174
email notification • 32
field initialization • 25
field validation • 25
JSP file • 163

L

language identifiers, ISO 639 • 162
locale
resource bundles • 163
retrieving for current user • 162
localization
event descriptions • 156
exception messages • 79, 161
extracting localized text • 163
identifiers • 162, 163
IMSExceptions.properties • 161
IMSResources bundle • 163
new skins • 163
strings • 163
Localizer class • 163
Logical Attribute API
called by CA Identity Manager • 46, 50
calling sequence • 50
compiling • 173
data persistence • 48
objects • 48
option list • 59
purpose • 46
sample • 51
self-registration extension • 64
use cases • 47
logical attribute handlers
configuring • 52
executed before business logic task • 50, 77
extended use case example • 54
object type • 41
purpose • 42
sample • 51
sharing data with other objects • 48
logical attributes
about • 39
adding to task screens • 39, 55
alternative validation methods • 46
elements of • 41
extended use case example • 54
fields and • 39
name • 53
physical attributes and • 40
predefined • 43, 133
purpose • 42
referencing in managed objects • 143
regular expression validation • 46
summary • 52
LogicalAttributeAdapter class • 48
LogicalAttributeContext interface • 48

M

managed objects
about • 139
accessing data in • 144
accessing through events • 153, 154
action type • 168
and event listeners • 94
attribute names • 143
attributes • 22, 143, 144
business logic task handlers • 80
committing changes • 144
integration with third-party applications • 125
interfaces • 140, 142
modifying in a task session • 150
permissions • 148, 149, 164
provider access • 144
read/write access • 144
referencing attributes • 143

retrieving attributes • 143, 154
retrieving through events • 153, 154
setting attributes • 143
specifying attributes to include • 148
subject of a task • 150
superinterfaces • 142
task session • 71, 150
task session access • 152
type • 168
validation of changes • 170
ManagedObject interface • 142
membership
constraints • 167
policies • 164
roles • 142
messages
localization • 161
storing exceptions • 161
ModifiableObject interface • 142, 149
modifyObject() • 142, 144, 146, 149
modifyObject(Task[]) • 140, 149
Multi-Select • 60

N

N search depth type • 168
NamedObject interface • 142
Notification Rule API
access to managed objects • 144
called by CA Identity Manager • 133
compiling • 173
objects • 134
sample • 136
task example • 32
use cases • 134
notification rules
about • 133
configuring • 136
predefined • 43, 133
sample • 136
NotificationRuleAdapter class • 134
NotificationRuleContext interface • 134, 153

O

object type • 168
logical attribute handler • 41
objects
events • 153, 157
subject of a task • 150
types • 168
ObjectType class • 168
oldPassword logical attribute • 43
OperatorType class • 168
option list
about • 59
adding to a task screen • 60
configuring • 60, 61
populating • 60, 62
source of list items • 60
order of execution
global and task-specific task handlers • 77
organization events • 157
Organization interface • 140
Organization Selector • 63
OrganizationEvent interface • 154
organizations
constraints • 167
identifier • 128
managed object • 140
parent • 154
specifying, in self registration • 62
OrgResolverAdapter obsolete • 129
OrgResolverContext obsolete • 129
OrgSelector class • 64
OrgSelectorAdapter class • 63, 64

P

ParentOrganizationEvent interface • 154
Participant Resolver API
access to managed objects • 144
calling sequence • 130
compiling • 174
objects • 129
sample • 130
task example • 32
participant resolvers
about • 127
configuring • 131
custom • 128
finding workflow approvers • 127, 130
order of precedence • 130
organization names • 127
sample • 130
types • 130
ParticipantResolverAdapter class • 129
ParticipantResolverContext interface • 129, 153
participants

finding • 127, 130
See also participant resolvers, Participant Resol • 130
%PASSWORD% • 43
password policies • 140
PasswordCondition class • 163
passwordConfirm logical attribute • 43
PasswordMessageType class • 168
PasswordPolicy interface • 140
passwords
invalid • 163, 168
password policy managed object • 140
permissions • 148, 149, 164
PermissionType class • 168
persisting data between objects
Business Logic Task Handler AP • 75
Logical Attribute API • 48
Workflow API • 123
phases
asynchronous • 28, 29
overview • 24
synchronous • 24, 25
physical attributes
about • 39
adding to task screens • 39, 55
fields and • 39
logical attributes and • 40
referencing in managed objects • 143
where defined • 41
policy conditions • 164
populating an option list • 60, 62
Post requests • 176, 188, 189
predefined logical attributes • 43
primary object. See subject of a task • 150
profile • 139, 150
profile tabs • 150
PropertyDict interface • 165
ProviderAccessor interface • 149
managed object access • 144, 146, 148
providers
defined • 147
directory-level validation • 170
events not generated • 146
managed object access • 144
provisioning
eTrust Admin • 139, 140, 165
identity policies • 159
roles • 165
ProvisioningRole interface • 140
ProvisioningRoleEvent interface • 154

Q

question and answer logical attributes • 43
Questions logical attribute • 43

R

read and write permissions • 148, 149
read/write access • 144, 153
rejected() • 86, 91
relationships
about • 150
adding and removing • 152
ResultsContainer class • 166
tabs and • 150
resource bundles
base name • 162
custom • 162
deploying • 174
exceptions • 79, 161
IMSResources • 163
locale • 163
name format • 162
storing • 162
ResultsContainer class • 166
retrieving
managed object attributes • 143, 154
managed objects through events • 154
role assignment events • 157
Role-Based Access Control • 140
role events • 157
role grant events • 157
Role interface • 140
Role type participant resolver • 127
order of precedence • 130
RoleDisplayData interface • 166
RoleObjectQuery class • 167
roles
display data • 166
managed object • 140
membership • 142
subject's relationships • 150
tabs and • 150
rule-based assignments
identity policies • 159
provisioning • 165
rules
constraints • 167

policy conditions • 164

S

samples
Business Logic Task Handler API • 76
Logical Attribute API • 51
Notification Rule API • 136
Participant Resolver API • 130
self-registration handler • 66
Workflow API • 124
scope of a search • 167
scripts, workflow • 121
sharing data with custom objects • 123
search depth • 168
search expressions
conjunction type • 168
defining • 167
operator type • 168
search operations
API objects • 167
filters and constraints • 167
SearchCursor class • 167
SearchDepthType class • 168
SearchExpression class • 167
secondary events
generating • 144, 153
generating from workflow script • 123
security checks
available in a task session • 151
unavailable with providers • 146
Self Registration
specifying the organization • 62
Self Subscribing Handler • 43
self-registration events • 157
self-registration handlers
and logical attribute handlers • 63
coding • 64
purpose • 63
sample • 66
%SELF_SUBSCRIBING% • 43
session. See task session • 70
setAttribute... methods and attribute validation • 152
setting managed object attributes • 143
Single-Select • 60
SiteMinder Directory • 139
skins, localizing • 163
SOAP document • 176, 178, 184, 187, 192, 194, 198
StringRef • 170
subject of a task
about • 150
modifying • 152
modifying in a task session • 152
relationships • 150, 166
tabs and • 150
submitting a task • 161
support
support, contacting • 3
synchronous phase • 24, 25, 34

T

task events • 157
Task Execution Web Service
configuring • 181
development overview • 184
exceptions • 187, 198
generated files • 196
operation types • 179, 180
overview • 176
Task.FieldId class • 168
Task interface • 140
task-level validation • 170
task screen
attribute validation • 170
attributes and • 39
business logic task handler and • 150
data flow • 42
exception messages • 79, 161
field validation • 170
localized labels • 163
logical attributes • 25, 53
option list • 60
tabs • 150
task session
about • 70
attribute validation • 170
contents • 150
event generation • 151
field validation • 170
managed objects • 71, 150
modifying objects in • 150
overview • 24
TaskObjectQuery class • 167
tasks
associating with a role • 140, 149
events in • 85, 153

exceptions • 187, 198
execution phases • 24
execution summary • 32
in asynchronous operations • 29
in synchronous operations • 25
localized names • 163
managed object • 140
operation flow • 180
operation types • 179, 180
overview • 24
POST request • 176
remote • 176, 177
session • 70
SOAP • 176, 178, 184, 187, 192, 194, 198
subject of • 150, 166
submitting • 161
workflow controlled • 29
task-specific business logic task handlers
order of execution; • 77
about • 73
TaskValidator interface • 170
TaskValidator.StringRef class • 170
technical support
technical support, contacting • 3
templates for email • 133
third-party applications
external tasks • 154
integration with • 125
toLogical() • 50
toPhysical() • 25, 50
trigger rules. See policy conditions • 164
TSContext interface • 168

U

user certification events • 157
user directory • 39
user events • 157
User interface • 140
USER notification rule • 133
USER object type • 43
USER_MANAGER notification rule • 133
user-defined data
option lists • 60
UserEvent interface • 154
users
admin overview • 24
managed object • 140
self-registered • 62

V

validate() • 25, 50
validation
directory-level • 170
task-level • 170
validation rules
on attributes • 152
types of • 170
VerifyQuestion logical attribute • 43
ViewUserQuery • 178
ViewUserQueryResult • 178

W

web service • 176, 184, 187, 194
web.xml file • 190
well-known attributes
mapped to logical attributes • 41
mapped to physical attributes • 41
referencing in managed objects • 143
workflow
approval task • 22, 129
event generation • 123, 151
scripts • 121
workflow activities
finding participants • 130
participants • 127
Workflow API
access to managed objects • 144
and compilation • 173
called by workflow scripts • 121
data persistence • 123
generating secondary events • 123
integration with third-party applications • 125
objects • 123
sample • 124
task example • 32
use cases • 122
WorkflowContext interface • 123, 153
WSDL • 176, 184