Sun Java™ System

# Identity Manager 6.0 Workflows, Forms, and Views

2005Q4M3

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054
U.S.A.

Part No: 819-4485-10

# Contents

## Workflow Services

## Identity Manager Forms

## XPRESS Language



## XML Object Language

## HTML Display Components



## Form and Process Mappings

# Preface

This guide provides detailed information about Sun Java™ System Identity Manager workflows, forms, and views — including information about the tools you need to customize these objects.

## How to Find Information in this Guide

The guide is organized in these sections:

- Chapter 1. Identity Manager Workflow — Describes the Sun Java™ System Identity Manager (Identity Manager) workflow and provides information helpful when using the Business Process Editor to create and edit workflow tasks.
- Chapter 2. Workflow Services — Describes the Sun Java™ System Identity Manager workflow services that are available to implementers who modify or create custom workflows.
- Chapter 3. Identity Manager Forms — Describes how to customize the appearance and behavior of selected pages in Identity Manager Administrator and User Interfaces by customizing the *forms* that define these pages.
- Chapter 4. FormUtil Methods — Describes the most commonly used FormUtil methods.
- Chapter 5. XPRESS Language — Introduces the basic features of XPRESS, an XML-based expression and scripting language used throughout Identity Manager.
- Chapter 6. XML Object Language — Introduces the basic features of the XML Object language, which is a collection of XML elements that you can use to represent common Java objects such as strings, lists, and maps.
- Chapter 7. Identity Manager Views — Introduces Identity Manager views, which are data structures used in Identity Manager.
- Chapter 8. HTML Display Components — Describes the Identity Manager HTML display component library. HTML display components are used when customizing forms.
- Appendix A. Form and Process Mappings — Lists the forms and workflow processes used in Identity Manager and their corresponding system names.

# Related Documentation and Help

Sun Microsystems provides additional printed and online documentation and information to help you install, use, and configure Identity Manager:

- *Identity Manager Installation*

  Step-by-step instructions and reference information to help you install and configure Identity Manager and associated software.

- *Identity Manager Upgrade*

  Step-by-step instructions and reference information to help you upgrade and configure Identity Manager and associated software.

- *Identity Manager Administration*

  Procedures, tutorials, and examples that describe how to use Identity Manager to provide secure user access to your enterprise information systems.

- *Identity Manager Technical Deployment Overview*

  Conceptual overview of the Identity Manager product (including object architectures) with an introduction to basic product components.

- *Identity Manager Workflows, Forms, and Views*

  Reference and procedural information that describes how to use the Identity Manager workflows, forms, and views — including information about the tools you need to customize these objects.

- *Identity Manager Deployment Tools*

  Reference and procedural information that describes how to use different Identity Manager deployment tools; including rules and rules libraries, common tasks and processes, dictionary support, and the SOAP-based Web service interface provided by the Identity Manager server.

- *Identity Manager Technical Reference*

  Reference and procedural information that describes how to load and synchronize account information from a resource into Sun Java™ System Identity Manager.

- *Identity Manager Audit Logging*

  Reference and procedural information that describes how to load and synchronize account information from a resource into Sun Java™ System Identity Manager.

- *Identity Manager Tuning, Troubleshooting, and Error Messages*

  Reference and procedural information that describes Identity Manager error messages and exceptions, and provide instructions for tracing and troubleshooting problems you might encounter as you work.

- *Identity Manager Help*

  Online guidance and information that offer complete procedural, reference, and terminology information about Identity Manager. You can access help by clicking the Help link from the Identity Manager menu bar. Guidance (field-specific information) is available on key fields.

# Product Support

If you have problems with Identity Manager, contact customer support using one of the following mechanisms:

- The online support web site at http://www.sun.com/service/online/us
- The telephone dispatch number associated with your maintenance contract

# We'd Like to Hear from You!

We would like to know what you think of this guide and other documentation provided with Identity Manager. If you have feedback - positive or negative - about your experiences using this product and documentation, please send us a note:

Sun Microsystems.
5300 Riata Park Court
Austin, TX 78727
Attn: Identity Manager Information Development

Email: idm-idd@sun.com

## We'd Like to Hear from You!

# 1 Identity Manager Workflow

This chapter describes Sun Java™ System Identity Manager workflow and provides information helpful when using the Business Process Editor to create and edit workflow tasks.

## Topics in this Chapter

In this chapter, you will learn about:

- Identity Manager workflow concepts
- Identity Manager Business Process Editor
- Workflow Toolbox

## Related Chapters

See these related chapters:

- *Workflow Services* – Describes the workflow service methods that are available to the Identity Manager implementer who is customizing Identity Manager.
- *XPRESS Language* – Lists and describes use expressions written in the XPRESS language to include logic in workflows and forms.
- *Introduction to the Business Process Editor* – Introduces the Business Process Editor (BPE) and describes how to start the tool, set editor options, and save the processes you have edited. You can also use the BPE to debug workflows and forms.

# Understanding Workflow

Identity Manager *workflow* defines a sequence of actions and tasks that are performed consistently according to a defined rule set. Using the Identity Manager Business Process Editor graphical interface, you can customize each workflow launched by the Identity Manager Administrator interface.

Before working with workflow, develop an understanding of:

- General workflow concepts
- How workflow is used in Identity Manager

# What is Workflow?

In general terms, a *workflow* is a logical, repeatable process during which documents, information, or tasks are passed from one participant to another for action, according to a set of procedural rules. A *participant* is a person, machine, or both.

In Identity Manager, this concept is specifically implemented as the Identity Manager workflow capability, which comprises multiple processes (workflows) that control creation, update, enabling, disabling, and deletion of user accounts.

Most of the Identity Manager tasks you perform are defined as a set of workflow processes. When you create a user in Identity Manager, for example, the corresponding workflow process defines and conducts activities that:

- Check password policies
- Send email to approvers
- Evaluate the results of each approval
- Create user accounts
- Create audit records

The Identity Manager Business Process Editor (BPE) allows you to customize each of these activities.

Workflows are typically launched as a side effect of checking in a view. Views are checked in when you click **Save** on a page that implements forms and views.

## Workflow Processes

Depending upon where you are in the product interface, workflows are referred as *workflows*, *tasks*, or *TaskDefinitions*.

### Task Definitions

From the Debug page of the Identity Manager Administrator interface, access workflow processes by looking at the list of `TaskDefinitions`. Select TaskDefinition from the Type menu adjacent to the **List Objects** button, then click **List Objects**. Identity Manager displays a list of the TaskDefinitions you have access to. When you customize a workflow, you are editing its `TaskDefinition` object.

Typically, when you edit and save a workflow definition, Identity Manager upgrades its `TaskDefinition` by:

- Renaming the repository version
- Adding the date of upgrade
- Saving the modified TaskDefinition without specifying the ID parameter.

The launched instance of a `TaskDefinition` is represented as a `TaskInstance` object, which you can also view from the Debug page.

### Task Definition Parameters

The following table lists the standard configuration parameters.

| Parameter | Description |
|---|---|
| `name` | Specifies the user-supplied name of the workflow as presented in the Identity Manager interface. Names should be unique among objects of this type, but objects of different types can have the same name. |
| `taskType` | Identifies the class that implements the TaskExecutor interface. For workflow, this value is Workflow. |
| `executor` | Identifies the name of the class that implements the task. By default, for workflows this class is com.waveset.workflow.WorkflowExecutor. |
| `suspendedable` | (Boolean) Indicates that the task can be suspended and resumed. Default is true. |
| `syncControlAllowed` | (Boolean) Indicates whether the user is permitted to request synchronous or asynchronous operation. Default is true. |

| Parameter | Description |
|---|---|
| `execMode` | Specifies the type of execution we should use by default. Default is sync.<br>If this value is null, or set to ExecMode.DEFAULT, we treat it as ExecMode.ASYNC. |
| `executionLimit` | Specifies the limit in seconds that the task is allowed to execute. The task can specify a limit on the amount of time it is allowed to execute. If it exceeds this limit, the scheduler is allowed to terminate it. A limit of zero means there is no limit.<br>Default is 0. |
| `resultLimit` | Specifies the limit in seconds that a task instance is allowed to live after the task has completed. Default is 0.<br>Once a task has completed or terminated, the TaskInstance containing the task result is typically kept in the repository for a designated period of time, after which it is automatically deleted.<br>0 – Indicates that the TaskInstance will be deleted immediately after the task is complete.<br>-1 – Indicates that the TaskInstance will never be automatically deleted, though it can be manually deleted by the user.<br>This parameter is typically set to a value that is equivalent to a few days for tasks that generate reports for later analysis. Set to zero for tasks that are run only for side effect and do not generate any meaningful result. |

| Parameter | Description |
|---|---|
| `resultOption` | (String) Specifies the options how the results of prior executions of repetitive tasks are handled. This object defines that data, and how to ask for it. Default is `delete`.<br>• `wait` – Prevents the task from being run until the old result is manually deleted or expires. If this is a non-scheduled task, it results in an error at the time it is launched. If this is a scheduled task, the scheduler simply ignores it.<br>• `delete` – Automatically deletes old results before executing the task. The old tasks must be in a finished state.<br>• `rename` – Renames old results before executing the task. The old task must be in a finished state.<br>• `terminate` – Terminates and deletes any currently executing task. This is similar to the DELETE option, but it also automatically terminates the task if it is running. |
| `visibility` | (String) Declares the visibility of this task definition. Default is `run schedule`. Other options include `invisible`, `run task`, and `schedule task`. |
| `progressInterval` | Specifies the interval in milliseconds that Identity Manager should check for progress updates.<br>The task can specify an interval at which the task will be updating its progress. Defaults to 5000 milliseconds (five seconds). Specifying a shorter interval will give you more current task status, but increases the load on the server. |

## About Manual Actions

A *manual action* is part of the workflow process definition that defines a manual interaction. When the workflow engine processes a manual action, a WorkItem object created in the repository. The work item must be marked "complete" before the workflow can proceed. Since most manual actions are used to solicit approvals, the work item table is under the Approvals tab.

Any manual action that belongs to a workflow is represented by a WorkItem objects in the repository.

### TaskInstances

Once a workflow task is launched, the workflow engine creates a *TaskInstance* in the repository. A TaskInstance is an object in the repository that holds the runtime state of an executing workflow process. It stores context variables and immediate transition information for the TaskDefinition from which it was spawned.

The TaskInstance references the descriptive TaskDefinition object through the TaskDefinition object's generated ID. If you edit a TaskDefinition, TaskInstances already in execution will continue to use the old TaskDefinition object, but new ones will use the modified TaskDefinition with its newly generated ID.

### When Are Task Instances Deleted?

The life of a TaskInstance is determined by the `resultLimit` parameter. If the result limit is zero, the task will be deleted immediately. If it is positive, the value is the number of minutes that the TaskInstance is kept.

To delete a suspended workflow TaskInstance,

1. Click the Manage Tasks tab in the Identity Manager Administrator interface.
2. Select View All Tasks.
3. Select the suspended TaskInstance, then click **Terminate**.

## Workflows in the Repository

Within the Identity Manager repository, a workflow exists as a configuration object typically of Type WFProcess. (The single exception to this object definition is the Create User workflow, which is defined as a ProvisioningTask object.) The taskType is always `Workflow`.

**Note** Identity Manager does not lock the repository while a workflow is executing. This is because workflows can run for days, and the repository cannot remain unlocked for that long. However, Identity Manager prevents you from launching another update workflow on the same user.

## Workflow Engine

The workflow engine is a software service that provides the run-time execution for a workflow process. The functions provided by the workflow engine to support a workflow process include:

- Interpreting the process definition
- Creating process instances and managing their execution
- Navigating between activities and creating work items for their processing

# Workflow Components

Workflow processes represent the steps that must be followed to reach a desired goal. Processes define:

- Workflow activities
- Relationships between activities
- Criteria needed to start, advance, and complete a process

Each workflow process is defined by one or more of these components:

- **Activity** – Represents a single, logical step in the process.
- **Action** – Defines how an activity is accomplished. An action can be a simple expression evaluation or a call to a complex Java class.
- **Transition** – Defines the movement from one activity to the next.
- **Split** – Defines the movement from a single activity to more than one activity. Splits are further defined as:
  - **OR Split** – Tests each transition path. The first path with a value of TRUE is taken.
  - **AND Split** – Takes each transition path.

**Join –** Defines the movement from multiple activities to a single activity. Join components are further defined as:

  - **OR Join –** Specifies that the first path to complete causes the next activity to begin.
  - **AND Join –** Specifies that the next activity cannot begin until all paths are complete.
- **Subprocess** – Defines a set of activities, actions, and transitions that can be called from other activities in the process.

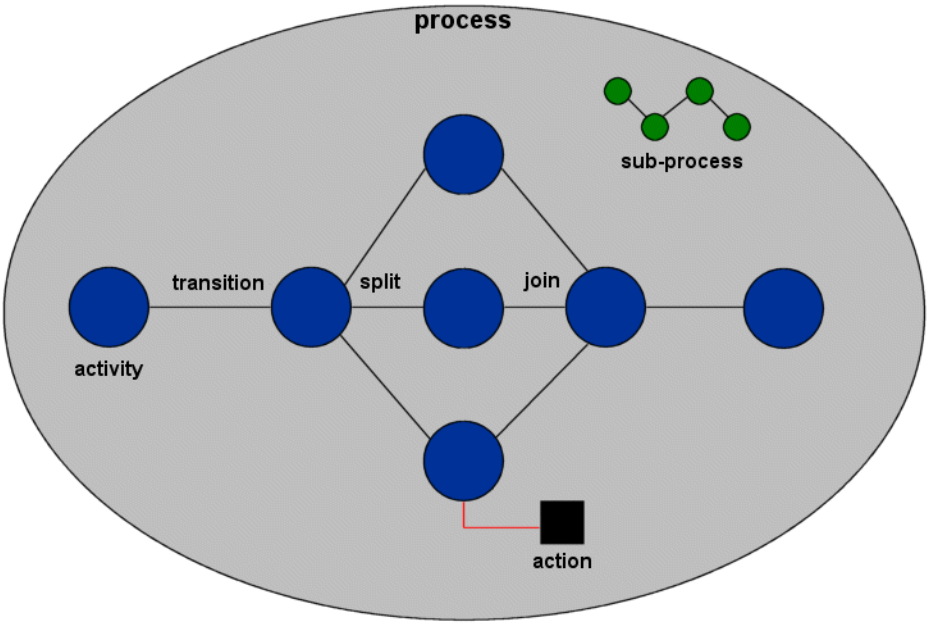



Figure 1. General Workflow Process and Components

# Default Workflow Processes

Using the BPE, you can edit the default Identity Manager processes to follow a custom set of steps. The Identity Manager workflow capability includes a library of default workflow processes, which includes:

- **User workflows** — Define the steps for tasks related to Identity Manager users, including creating, deleting, updating, enabling, disabling, and renaming users.
- **Identity Manager object workflows** — Define the steps for all tasks related to Identity Manager objects, including resources, resource groups, organizations, and organizational units. For example, some workflows, such as the Manage Role and Manage Workflows workflows simply commit view changes to the repository, while providing hooks for approvals and other customizations.
- **Miscellaneous workflows** — Define the steps for various Identity Manager features and objects such as reconciliation, Remedy templates, and other custom tasks.

# Using the Business Process Editor

The BPE application is a standalone, Swing-based Java application that provides a graphical and forms-based view of each process.

Follow these general steps when working with workflow in the BPE:

- Start the application
- Load a default workflow process
- Customize the workflow process
- Validate the process (this step iterates throughout the customization process)
- Debugging the process
- Save the process for use in Identity Manager

The following sections provide specific information and procedures for each of these general steps.

For guidelines on starting the BPE, setting editor options, using keyboard shortcuts, and using the BPE debugger, see *Introduction to the Business Process Editor*.

# Loading a Workflow Process

To load a workflow process:

1. Select **File → Open Repository Object** from the menu bar.

**Tip** You can also use the Ctrl-O shortcut. For a complete list of shortcuts you can use in the BPE, see *Introduction to the Business Process Editor*.

2. If prompted, enter the Identity Manager Configurator name and password in the displayed login dialog, and then click **Login**.

   The BPE displays a list of items to select, which include:

   - Workflow tasks 
   - Workflow subprocesses 
   - Email templates 
   - Forms 
   - Views 

- Rules 
- Libraries 
- Configuration Objects 
- Generic Objects 
- MetaViews 

**Note** Items displayed may vary for your Identity Manager implementation.

3. Select a workflow task or subprocess, and then click **OK**.

**Note** If you are loading a workflow process for the first time, the workflow components displayed in the right pane may not display correctly. Right-click in the right pane, and then select Layout to re-display the diagram.

# Getting Around in the Business Process Editor

Before you start customizing a workflow task, you should know how to work, view and enter information, and make selections in the BPE.

The BPE interface includes a menu bar and dialogs for selections. The primary display is divided into two panes: the *tree view* and the *diagram view*.

## Tree View

The tree view (in the left interface pane) shows a hierarchical view of a task, form, view, or rule. It lists each variable, activity, and subprocess in order, nesting actions and transitions under each activity.

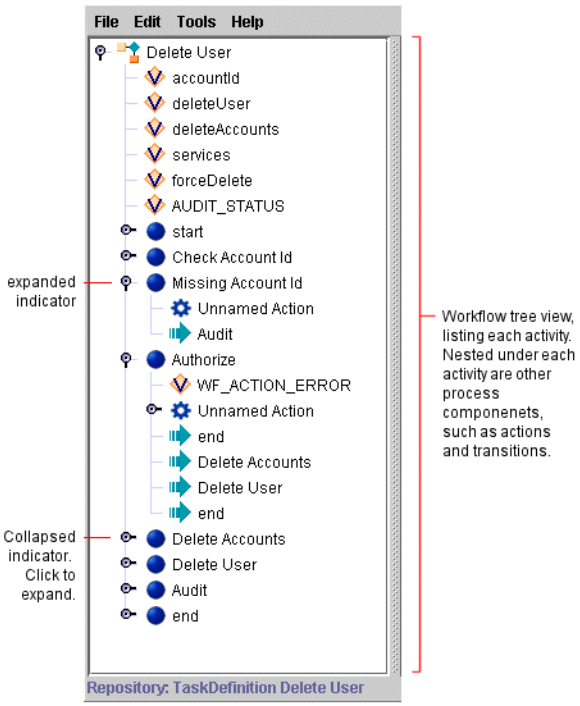



Figure 2. BPE Tree View

## Tree View Icons

Icons that display in the tree view represent the different elements of each task.

| This icon… | Represents this process element… |
|---|---|
|  | **activity** |
|  | **action** |
|  | **argument** |
|  | **result** |
|  | **return** |
|  | **subprocess** |
|  | **transition** |
|  | **variable** |

Table 1. Process Element Icons

Each of these elements is defined later in this chapter.

## Diagram View

The diagram view appears in the right interface pane, and provides a graphical representation of a process. Each icon represents a particular process activity.

### Repositioning Components

You can rearrange the components that display in the diagram view:

- Click and drag an icon to reposition it within the diagram.
- Right-click in the diagram view to display the actions menu, and then select **Layout** to automatically reposition all components in the diagram.

#### Notes

- Sun recommends that you use the **Layout** feature selectively. It is most useful when you first display a process. Often, the process does not appear completely when first displayed.
- You can set editor options so that layout is adjusted automatically when you open a workflow for the first time. For more information about setting editor options, see *Introduction to the Business Process Editor*.

To save your positioning for future editing sessions, select **File → Save to Repository**, if you want to preserve your revisions in the repository.

### Changing Displayed Connectors

You can select whether you want arrows to display on the lines connecting components, and the type of arrows to display. Right-click in the diagram view, and then select **Arrows**. Display options are **None**, **Open**, and **Filled**.

## Menu Selections

Use the menu bar or the action (right-click) menu to work in the interface. In the tree or diagram view, click to select an item, and then right-click to display action menu selections available for that item.

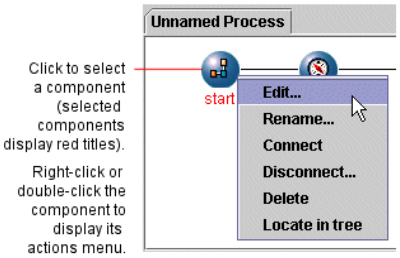



Figure 3. Diagram View Component Action Menu

## Workflow Dialogs

Each workflow component has an associated dialog that allows you to define the component and its actions. Some dialogs, such as the Activity dialog, displays tabs that access additional dialog pages.

Each dialog is described in detail in the following sections.

### Editing Dialog Fields (Changing Field Value Type)

Some dialog fields behave differently depending on the field value type you have selected. If the value type is String, you can type text directly into the field. If the value type is Expression, Rule, or Reference, click **Edit** to edit the value.

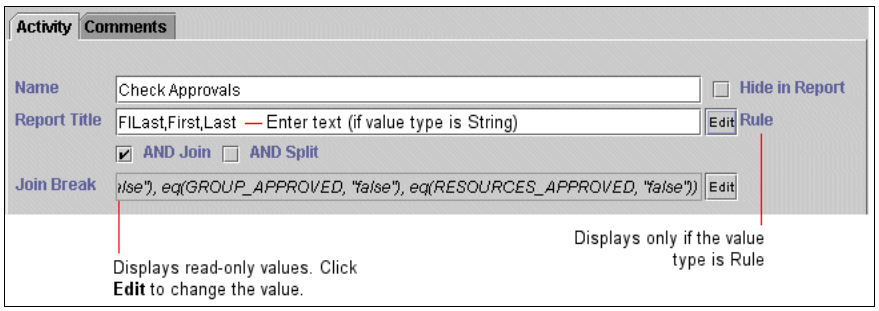



Figure 4. Edit Field

To change value type, click **Edit**, and then click **Change Type** (if the current value is String); or right-click and then select Change Type (if the current value is Expression, Rule, or Reference).

## Changing Display Type

In several dialogs, you can change the way information displays. For example, on the Script tab of the Actions dialog, right-click to change the default display type (Graphical) to an alternate display type.

Display types are:

- **XML** – Displays XPRESS or JavaScript source. You may prefer this display type if you are comfortable with XML.

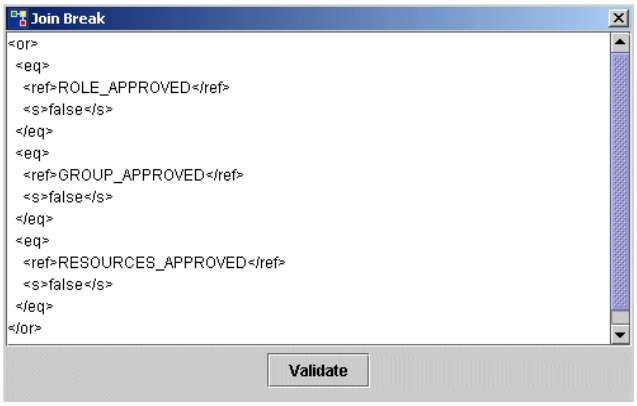



Figure 5. XML Display

- **Graphical** – Displays a tree of expression nodes. This display type provides a structure overview.

- **Property Sheet** – Displays a list of properties. Some of these can be edited directly. Others may require launching another dialog. For efficiency, use this display type when creating new expressions. (You can enter expression arguments more rapidly in this view than in the graphical view.)

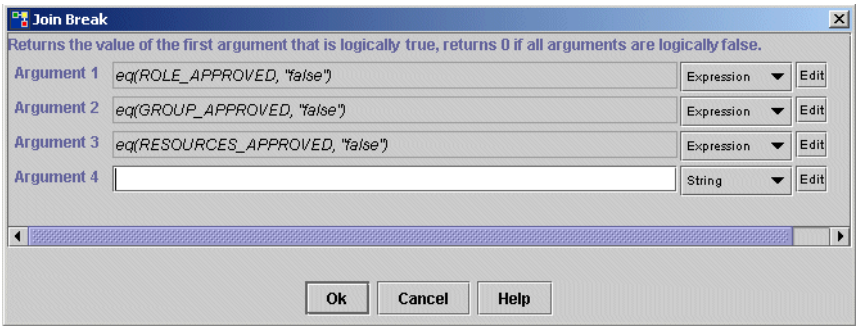



Figure 6. Property Sheet Display

- **Functional** – Displays a language similar to that used by popular spreadsheets. You might choose to use this display type if you are familiar with spreadsheet programming.

# Reviewing Process Summary Details

Double-click a process to view, at a glance, the variables used by the process and all process activities. Tabs that display on the Process dialog are:

- Process
- Repository
- Task Definition
- Comments

## Process Tab

Use to access TaskDefinition properties for this process (including the task name) and the Report Title. It also lets you re-order variables and activities for better visual organization. (Reordering in this list does not change interpretation of the process.)

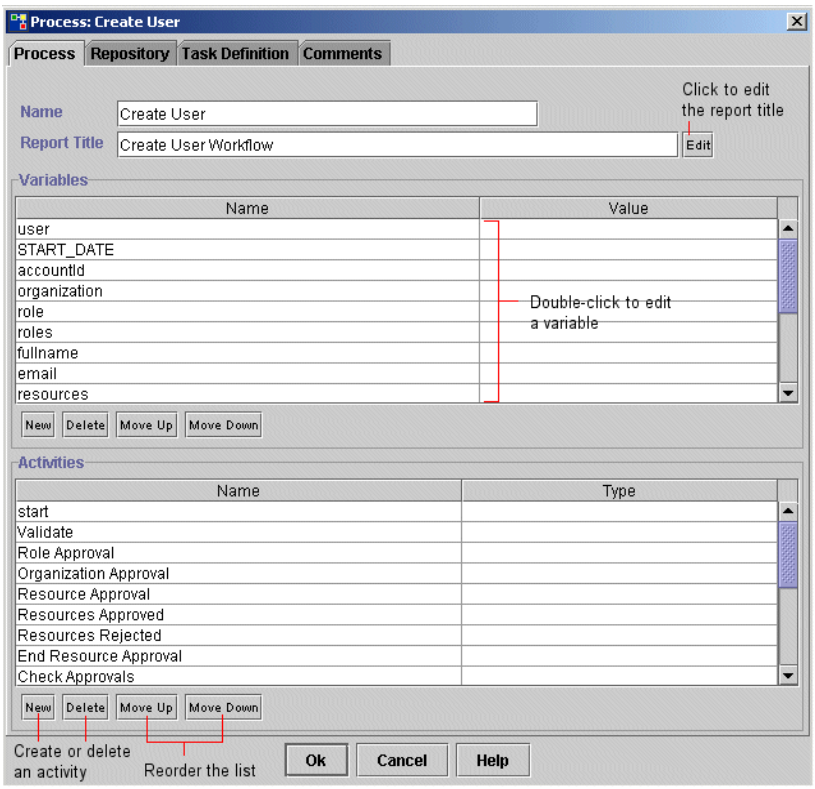



Figure 7. Process Dialog – Process Tab

## Repository Tab

Use to display read-only information about the process. You can also set the organization where the task is stored.

**Note** Generally, workflows reside in the Top organization since you need Configurator-level privileges to edit them.

You can also change the Process Type from Main Process to Sub Process. A Main Process corresponds to a TaskDefinition object in the repository, and may appear in the Run Tasks or Schedule Tasks list. A Sub Process is stored as a configuration object and does not appear in the Run Tasks or Schedule Tasks lists. The content of a subprocess can be any expression that computes either a String or an ObjectRef. Process names can be qualified as `TaskDefinition:`***your_task*** or simply ***your_task***. If unqualified, Identity Manager first looks for configuration objects before TaskDefinition objects.

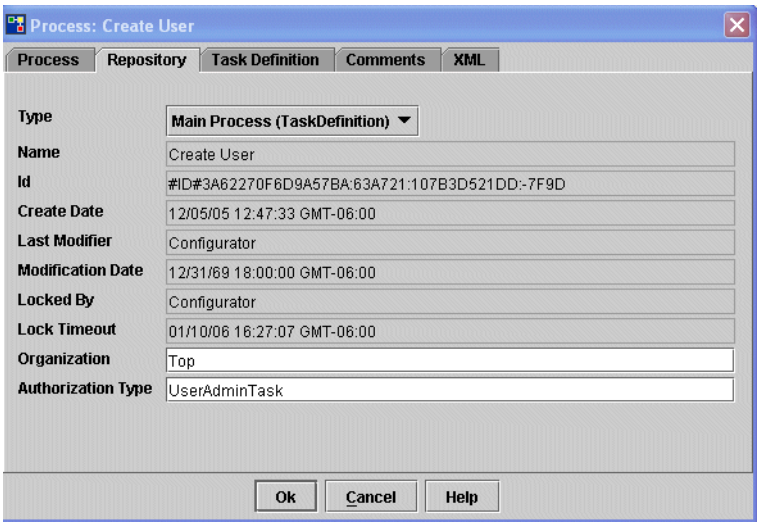



Figure 8. Process Dialog – Repository Tab

## Task Definition Tab

Displays if the selected process is defined as a TaskDefinition object. (Most default Workflow processes are defined as TaskDefinition objects.)

From this area, you can specify:

- Result limit – If non-negative, specifies the number of seconds that the system will retain results after a task has completed. The default value for provisioning tasks is 3600 seconds (`resultLimit='3600'`), or approximately one hour. Other tasks default to 0 seconds unless the task is changed to another value. When the result limit is 0, the task is cleaned up the next time the task scheduler wakes up after the task completes, which is typically a few seconds later.

  A negative value indicates the task instance will never be removed automatically. However, you can manually remove it.
- Result option – Specifies how the system treats the results of previous task executions. Options are:
  - Wait – Suspends a task.
  - Delete – Removes the task and its results from the system.
  - Rename – Renames the task.
  - Terminate – Stops a task in progress. The task cannot be restarted.

- Visibility – Specifies whether this task will appear in the Run Task or Schedule Task list in the Identity Manager Administrator interface. Options are:
  - Invisible – The task does not appear in the interface.
  - Run task – The task displays in the Run Task list.
  - Schedule task – The task displays in the Schedule Task list.
  - Run & schedule task – The task displays in both lists.

### XML Tab

Use to view and edit the raw XML for the process. You can validate changes before clicking **OK** to save. You can choose to edit the XML for the entire process to edit the individual activities using the diagram view.

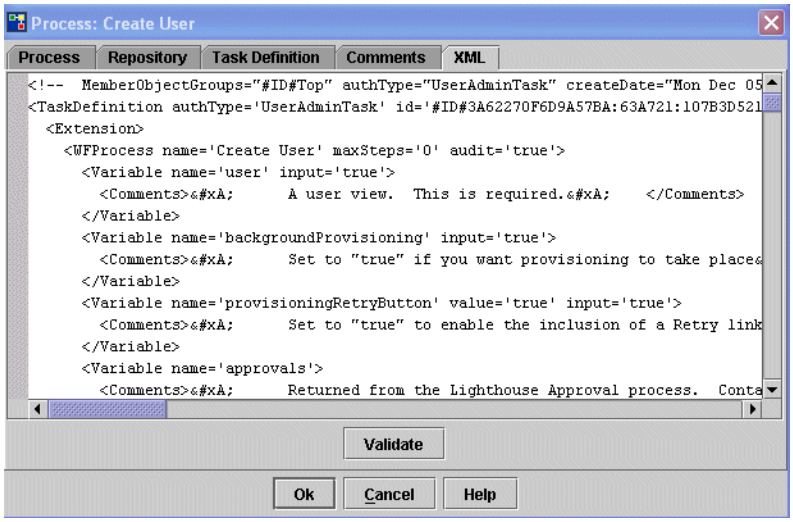



Figure 9. Process Dialog — XML Tab

## Defining Activities

An *activity* is a single step in the process. Each activity can contain multiple components, including transitions, variables, and actions.

Use the Activity dialog to define its components elements. Right-click the activity to display an action menu, and then select **Edit** to display its dialog.

# Activity Dialog

The Activity Dialog contains two tabs:

- Activity (default display)
- Comments

## Activity Tab

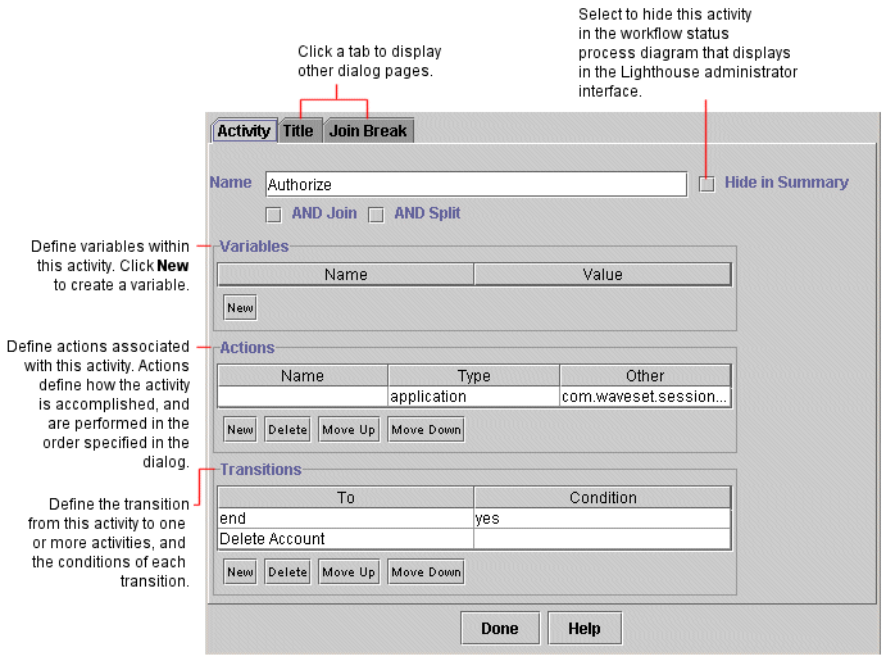



Figure 10. Activity Dialog: Activity Tab

Define and access these fields from the Activity tab:

- **Name** – Enter a name for the activity. This name displays in the BPE and, by default, in the workflow process report that displays in the Identity Manager Administrator interface. (You can set an alternative name to display in the Administrator interface in the Report Title field.)
- **Hide in Report** – Select to hide this activity in the Identity Manager Administrator interface workflow status process report. While Identity Manager administrators need to see how their requests are processed and their request status in the workflow process, they do not need to see unimportant activities.
- **Report Title** – Define the name that will display for this activity in the Identity Manager Administrator interface workflow status process report. A report title can be defined as a String, Expression, Reference, or Rule. Click **Edit** on the dialog to edit the title.
- **AND Join** – Select to specify that the activities that transition to this activity are defined as AND joins. If you do not select this option, joins are treated as OR joins.

  When you select this option, the dialog re-displays a field that allows you to define a join break, which forces an AND join to break if a certain condition is met.

  For example, a join break is used in the following workflow process. If any one of three approval types (role, organization, or resource) are required but rejected by an approver, the join break specifies that the process move to the Request Rejected activity (and then to End) as soon as the one approval type is rejected, rather than wait for all three to complete.

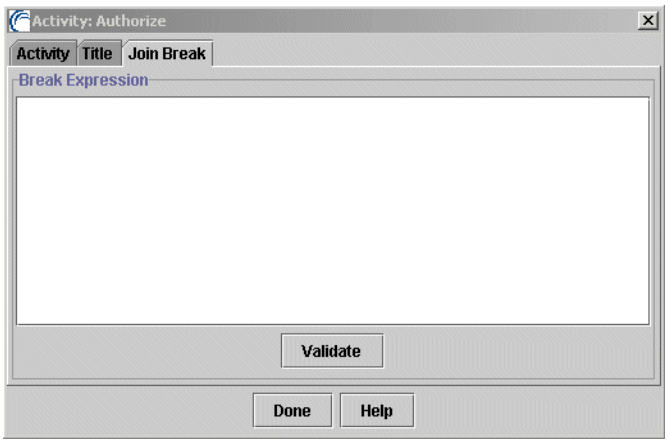



Figure 11. Activity Dialog: Join Break

You can define a join break as an Expression, Reference, or Rule. Click **Edit** on the dialog to edit the join break.

For help with join breaks defined as expression, refer to the detailed information about the XPRESS language.

- **AND Split** – Select to specify that the activities to which this activity transitions are AND splits. (The Transitions area defines the activities to which this one transitions.) If you do not select this option, splits are treated as OR splits.
- **Variables** – Defines one or more variables within this activity. Double-click a variable to define its name and value.
- **Actions** – Defines actions associated with this activity. Double-click an action to define its type and behavior.
- **Transitions** – Defines the transitions from this activity to one or more activities. Double-click an item in the list to define its behavior.

Joins, splits, variables, actions, and transitions are described in detail in the following sections.

## Specialized Activities: Start and End

Each process must contain activities that begin and end the process. These specialized activities are *start* and *end*.

###  Start Activity

The start activity should have no associated actions. It has one or more defined transitions to the next activity that begins the process.

###  End Activity

The end activity should have no associated actions or transitions. This ensures that the process can definitively end. The first process path that reaches this activity stops the process.

**Note** When the process ends, any outstanding manual actions are terminated. Outstanding application calls continue to completion, but results are ignored.

## Defining Variables

You can define several types of variables that are used within a process:

- **Process variables** – Accessible to all activities. Process variables defined within a process display in the BPE tree view.

  Any attribute associated with an Identity Manager user can be accessed through a single process variable named `user`. The value of this variable is the *user view*, a complex structure that contains all of the user attributes. You access attributes of the view by building path expressions such as `user.waveset.email`. See *Identity Manager Views* in this guide for more information.
- **Activity variables** – Defined within an activity. Activity variables are used solely within that activity.

  To create a variable from the Activity dialog, click **New**. To edit a variable, double-click a variable in the list.
- **Action variables** – Defined within an action. Action variables are used solely by the action where they are defined. To create a variable from the Action dialog, click **New** in the Variables area. To edit a variable, double-click a variable in the list.

### Editing Variable Details

On the Variable dialog, you can define values for the variable. Use the Value field to enter a simple string value as the initial variable value. This value can be changed by an action within the workflow. Alternatively, you can select a value type (such as Expression or Rule), and then click **Edit** to enter values.

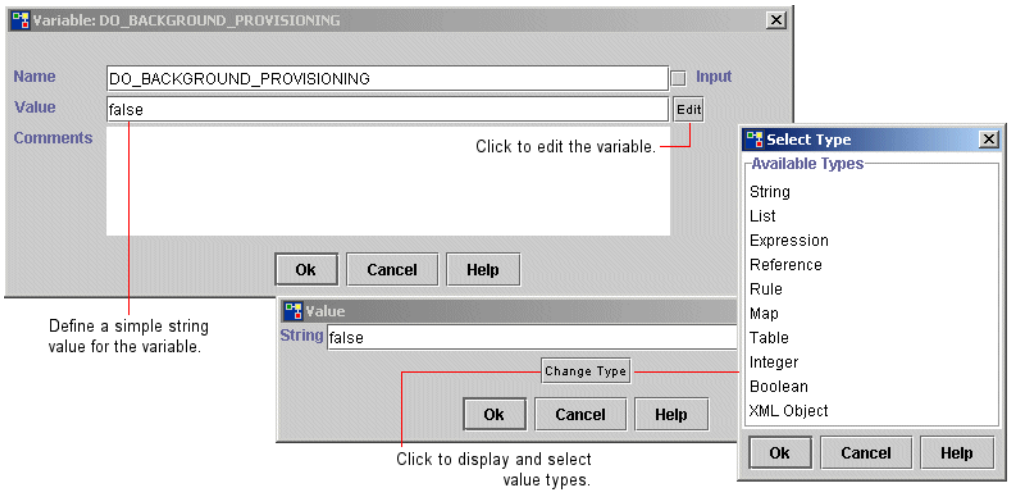



Figure 12. Variable Dialog

Value types include:

- String – Simple string constant.
- List – Static list, such as an XML object list. This type is infrequently used in workflows (but used occasionally in forms).
- Expression – Complex expression.
- Reference – Simple reference to a variable.
- Rule – Simple reference to a rule.
- Map – Static map, such as an XML object map. Used rarely.
- Integer – Integer constant. Can be used to make clearer the semantics of a value. Can be specified as String. The BPE coerces the string into the correct type.
- Boolean – Boolean constant. Can be used to make clearer the semantics of a value. Can be specified as String. The BPE coerces the string into the correct type. Boolean values are specified with the strings `true` and `false`.
- XML Object – Complex object that allows you to specify any of a number of complex objects with an XML representation. Examples include EncryptedData, Date, Message, TimePeriod, and WavesetResult.

## Defining Actions

Actions define the details of each activity. One activity can contain multiple actions, which are categorized as:

- **Script** – Typically, a short XPRESS or JavaScript program.
- **Application** – A call to a Java class that performs a specific action. These Java classes are registered with the workflow engine and are common to account management activities, such as sending email and creating a user.
- **Subprocess** – A call to another workflow process. The subprocess can be defined within the main process or a reference to an external process. The content of Subprocess can be any expression that computes either a String or an ObjectRef.
- **Manual** – An action that requires manual interaction before continuing to the next activity or activities.

Use the Activity dialog to manage actions. To create an action, click **New**. By default, actions are created as scripts.

## Action Dialog

The Action dialog contains five tabs:

- **Action** – Displays as the default dialog view. Define and access these fields from the Action tab:

- **Name** – Enter a name for the action. This name displays in the BPE.

  With the exception of manual actions, actions do not appear in the workflow process report that displays in the Identity Manager Administrator interface. You can select an alternative name to display in the Administrator interface for manual actions. (Do this in the Report Title field.)

  **Hide in Report** – Select to hide this activity in the Identity Manager Administrator interface workflow status process report.

  **Report Title** – Define the name that will display for this action in the Identity Manager Administrator interface workflow status process report. A report title can be defined as a String, Expression, Reference, or Rule. Click **Edit** on the dialog to edit the title.

  **Type** – Select the action type (Script, Application, Subprocess, or Manual).

  **Variables** – Define one or more variables for use in this activity.

  **Returns** – Assign the resulting values to variables that can be used later by this activity or a subsequent activity.

  **Results** – Set the Waveset Results values. These values appear on the Identity Manager Administrator interface.
- **Script** – Set up scripts to analyze and modify variables.
- **Conditional** – Set conditional statements for an action. For example, if an activity has multiple actions, you can define an expression that will cause one of the actions not to be performed if certain conditions are met.
- **Iteration** – Allows a single action to be called multiple times. Similar to a `for` loop.
- **Comments** – Enter information you want to remember about this action. Comments are not displayed in the process diagrams, but can be used to record information to assist workflow designers in understanding the action.

## Script

A script uses XPRESS logic to evaluate items in the process and to set values. Set up scripts on the Script tab of the Action dialog. Click **Validate** to check for logic errors in the script.

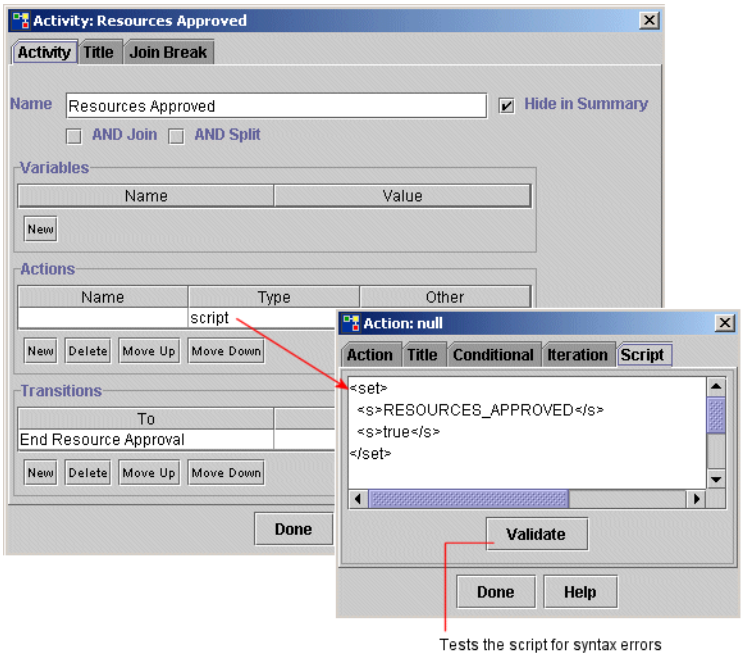



Figure 13. Script

For example, this process includes an activity called Resources Approved. The activity contains a script that sets the value of RESOURCES_APPROVED to true, which then transitions to the End Resource Approval activity.

## Application

An application supports processing for an activity. Applications use:

- **Variables** – Define local variables or set the values of variables passed in to the application.
- **Arguments** – Describe the data passed in to the application.
- **Returns** – Assign the resulting values to variables that can be used later.
- **Results** – Set the task result values. These values appear on the Identity Manager Administrator interface.

You can register your own Java methods for use in the BPE. For more information, see *Adding Applications*.

## Subprocess

Subprocesses define activities, actions, and transitions that can be called from other activities in the process. A *subprocess* can be defined within the main process or a referenced from an external process. The content of subprocess can also be any expression that computes either a String or an ObjectRef. Its name can be dynamically calculated by a rule. A subprocess has its own defined start and end.

Often, subprocesses are defined and used for a set of steps that must be repeated. When you call a subprocess (in the Actions area of the Activity dialog), you pass it arguments that it processes. It then returns values that you can assign to new variables.

In the following example workflow, the Role Approval activity calls a subprocess called approval-process. This subprocess:

1. Checks to see if there are approvers assigned to role and organization approval
2. Sends notification to the approver that there is a request pending
3. Waits until the approval is received
4. Returns each approver's response

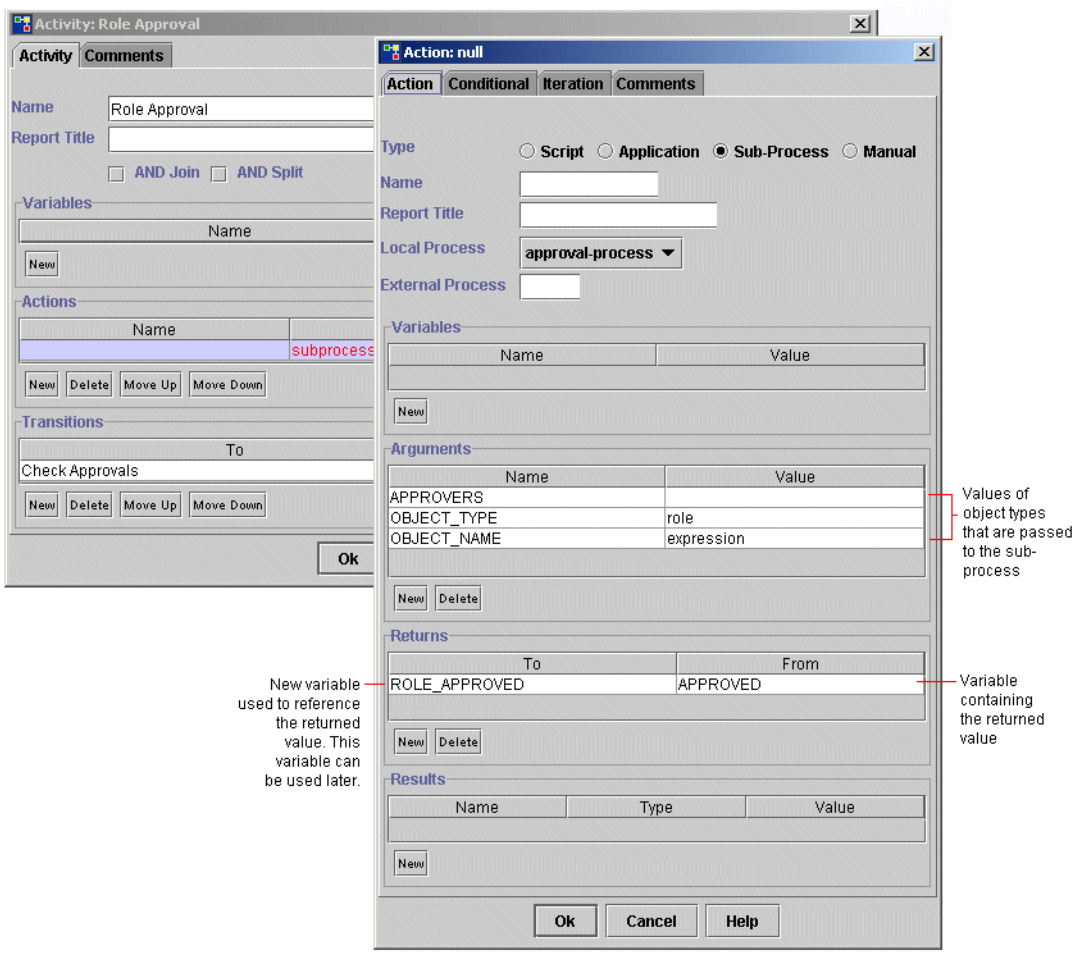



Figure 14. Subprocess (Create User Workflow)

## Manual

*Manual actions* are process steps that cannot be automated. They require human interaction for a process to continue.

Manual actions must have an owner (the person who is required to interact with the action). The action owner can be an Identity Manager administrator or user. You can specify an owner or define an expression that determines the owner.

In the following example, the approval process incorporates a manual action called Approve, which requires that the manual action owner (defined in the Owner field) indicate a start date for the user.

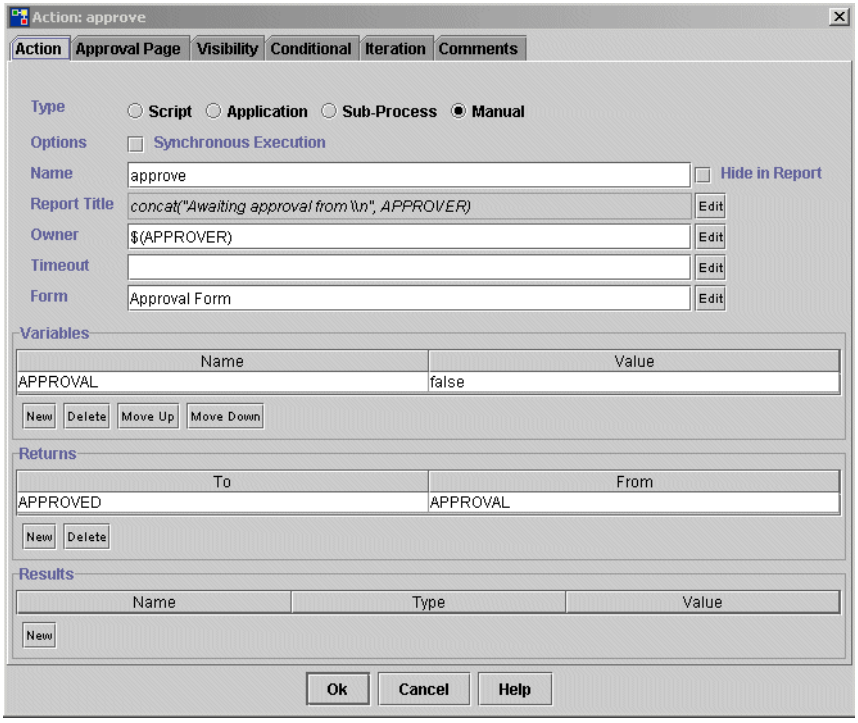



Figure 15. Manual Action (Standard Approval Workflow)

## Specifying an Owner

When specifying a manual action owner, specify Identity Manager administrator owners in the form `$(`*Owner)*. Specify Identity Manager user owners in the form `User:$(`*Owner*`)`.

## Specifying a Timeout

Use the Timeout field to specify a maximum amount of time that the system will wait for an administrator to complete the action. This capability is useful, for example, when implementing an escalation process whereby a request is forwarded to an alternate approver after a specified time. If the Timeout value is reached, the activity will stop, then proceed down the defined escalation path. You can find out the timeout value on the default Create User page, where it occurs in the subprocesses of the main process. Alternatively, you can create an Escalation Timeout rule.

You can enter a timeout number as a literal value, or it can be computed by an expression. If the number is positive, it indicates minutes. If negative, it indicates seconds.

If you specify a timeout for a manual action, you would typically check in your workflow to confirm that the manual action completed. You would examine the workflow because the responses, or transitions, in your workflow differs based on whether someone approved the action or whether the action timed out. In the former case, the workflow would move forward. If the latter, you would likely receive another notification and/or escalation. The workflow variable WF_ACTION_TIMEOUT will be set to `true` if the person did not respond in time.

**Note** You can also create a rule to set the timeout value. To do this:

1. Select **File → New → Rule** from the menu bar.
2. In the New Rule dialog, enter a name for the timeout rule.
3. In the Arguments area, click **New**.
4. In the argument dialog, enter a timeout value in the Value field, in the form:

   `<i>`***Value***`</i>`

The Form field specifies the form to present to the action owner. As an alternative to a specific form name, you could also could define an expression that logically determines the name of the form.

## Example

The following Create User workflow has been modified to call an escalate activity if a timeout value is reached. If the timeout is not reached, then the results of the APPROVED variable are evaluated. The results of the evaluation determines whether to transition to the approved or rejected activity.

```
<Activity name='Wait'>

   <ManualAction name='approve' timeout='180'>
   <Owner name='$(APPROVER)'/>
   <Variable name='APPROVAL' value='false'/>
   <Return from='APPROVAL' to='APPROVED'/>
   <FormRef>
     <ObjectRef type='UserForm' id='#ID#UserForm:ApprovalForm'/>
   </FormRef>
   <ReportTitle>
     <concat>
       <s>Awaiting approval from \n</s>
       <ref>APPROVER</ref>
     </concat>
   </ReportTitle>
   </ManualAction>



   <Transition to='Escalate'>
```

```
      <eq>
        <ref>WF_ACTION_TIMEOUT</ref>
        <s>true</s>
      </eq>
    </Transition>

    <Transition to='Approved'>
      <eq>
        <ref>APPROVED</ref>
        <s>true</s>
      </eq>
    </Transition>

    <Transition to='Rejected'/>

</Activity>
```

## WorkItem Types

Manual actions now have the ability to assign a type to the work item that is generated when the manual action is executed by the workflow engine. You can assign the work item type in a customization to filter the set of work items to be displayed or operated upon.

The following work item types are recognized by the system.

| Work Item Type | Description |
|---|---|
| approval | Indicates that the work item represents an approval. |
| wizard | Indicates that the work item represents an arbitrary interaction with the user. |
| suspend | Indicates that the work item is temporary. Use this type to force a workflow into background execution. |

In addition, you can assign customized work item types. For example, you might set the work item type to `resource` to represent a resource approval and `role` to represent a role approval.

### Authorization Types

Manual actions can also specify the *authorization type* of the WorkItem to be created. The authorization type differs from the item type in that the system automatically filters the work items returned in a query to exclude those for which the current administrator is not authorized. Typically, any administrator with the Approver capability is authorized to view all work items in the organizations they control.

To specify a work item authorization type in the manual action, use the `authType` attribute as follows:

```
<ManualAction authType='RoleApproval'>
```

### Assigning WorkItem Types

To specify an item type in the ManualAction definition, set the `itemType` attribute as shown in this example:

```
<ManualAction itemType='approval'>
```

### Restricting Administrative View Capabilities for WorkItems

Typically, any administrator with the Approver capability is authorized to view *all* work items in the organizations they control. If you want an administrator to view only a subset of the work items in an organization, follow these steps:

1. Define new authorization types that extend the WorkItem type. For example, define the `RoleApproval` type.
2. Define new capabilities that have rights on the new authorization types rather than WorkItem itself. For example, define a Role Approver capability that has rights on the `RoleApproval` type.
3. Assign the Role Approver capability to an administrator rather than the general Approver capability
4. Set appropriate authorization types in each manual action in your workflows.

## Creating Transitions

Transitions define the rules by which an activity moves to one or more other activities. A *transition* can be conditional, which means that it will be taken only if certain conditions are met.

Simple activities can contain only one unconditional transition that is taken as soon as the actions within the activity are complete.

An activity with more than one transition is a *split*. An activity with more than one incoming transition is a *join*.

## Split

A *split* is an activity that contains two or more transitions. A split lists two or more activities in the Transitions area.

In an OR split, each condition defined for the transition is evaluated. Transitions are tested in the order listed. The first path to return TRUE is taken. With an OR split, one path (the default path) must not have any conditions associated with it. An OR split is the default split when you create two or more transitions in an activity.

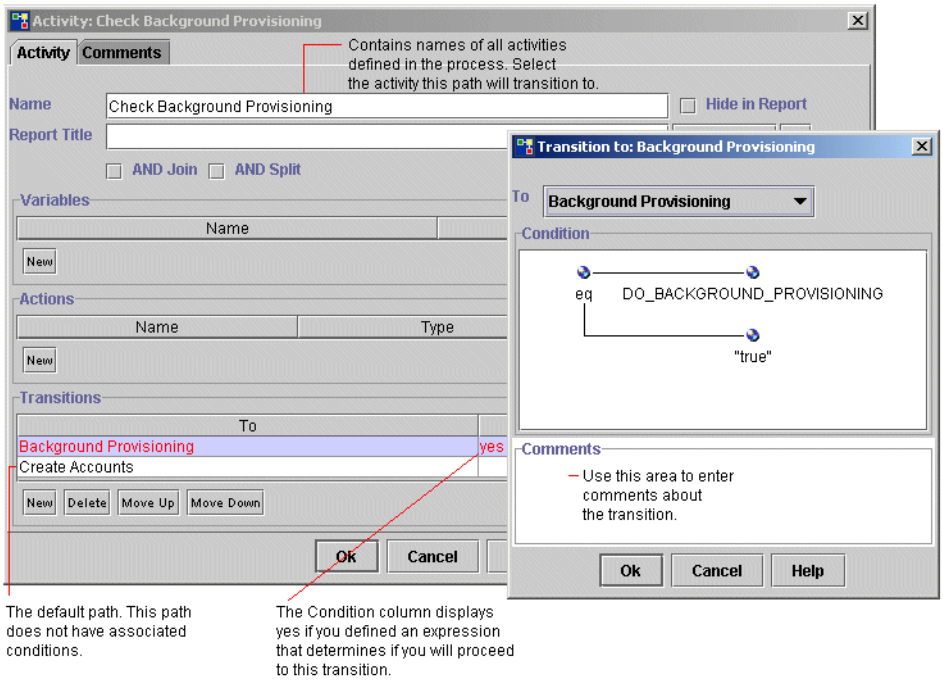



Figure 16. OR Split

In an AND split, each transition path listed in the Transitions area is taken. Select AND splits on the Activity dialog.

To add a transition, click **New** (in the Activity dialog Transitions area), and then select a component from the list of options.

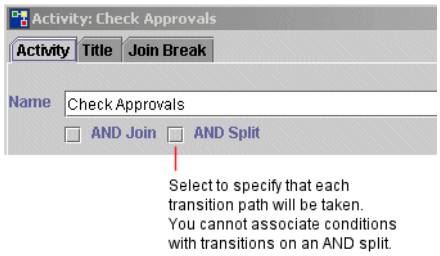



Figure 17. AND Split

## Join

A *join* is an activity to which two or more activities transition. You do not explicitly designate an activity as a join. The transitions defined in the preceding activities define it.

By default, joins are OR joins, meaning that the first path that reaches this activity causes the next step to proceed.

You can designate an activity as an AND join. AND joins require that all paths to it complete before the next step will proceed.

In the following example process, the Check Approvals activity is an AND join. Role Approval, Organization Approval, and End Resource Approval activities all must complete before the next steps can be taken.

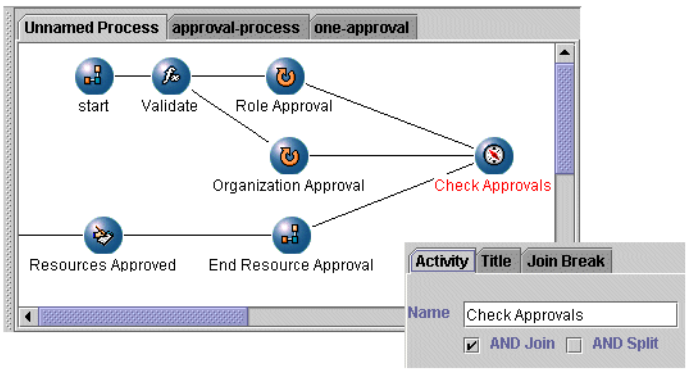



Figure 18. AND Join (Create User Process Check Approvals Activity)

### Join Breaks

A special condition on joins are *join breaks*, which cause an AND join to complete if a certain condition is met.

In the following example process, the Check Approvals activity contains a join break that allows the activity to complete if any one of the Role Approval, Organization Approval, or End Resource Approval activities returns a value of FALSE. In this case, it is not the completion of a step that causes the activity to continue, but a specific condition that is met.

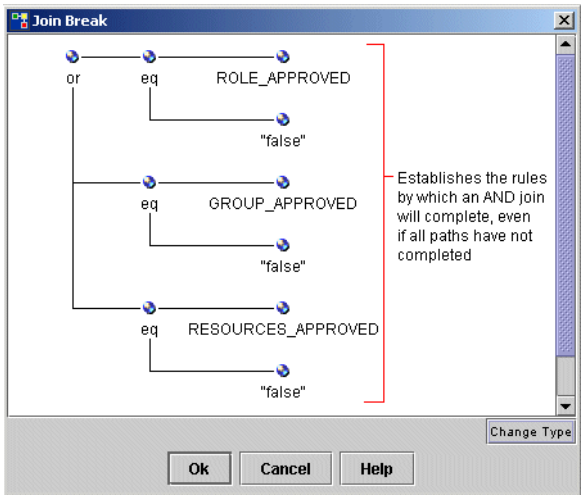



Figure 19. Join Break (Create User Process Check Approvals Activity)

# Updating a Process for Identity Manager Use

If you customize a process, validate and save your changes to ensure that the workflow process completes correctly and as you expect. After saving, import the modified workflow for use in Identity Manager. You can also use the BPE debugger.

## Editing a Workflow in Production

Do not customize a workflow process in a production environment.

Problems can emerge if you edit workflow activities or actions while instances of the original workflow are running. Specifically, the TaskInstance contains a reference to the workflow TaskDefinition and stores the current activity or action by ID. Changing these IDs may prevent the task from restarting where expected when execution resumes.

If you cannot avoid editing a workflow in a production environment, use the following procedure. It will help prevent the loss of pending work items from task instances that are using the old definition.

1. Rename the current TaskDefinition to include a time stamp. For example, to modify the Create User workflow, rename the TaskDefinition from Create User to Create User 20030701. You can rename a workflow TaskDefinition with the BPE.
2. Save and import the edited workflow.

Following this procedure will help prevent problems with existing Create User tasks that may be in a suspended state within Identity Manager. This allows the TaskDefinition to keep its unique ID, which is referenced inside suspended tasks.

# Saving

To save your changes to a workflow process and check it into the repository, select **File → Save in Repository** from the menu bar.

**Note** You can use **File → Save As File** to save the workflow as an XML text file. Save the file in the form *Filename*`.xml`.

# Validating

You can validate changes to workflow processes at different stages of the customization process:

- If you are working with XML display values, when adding or customizing variables, activities, actions, or transitions, click **Validate** to validate each change to the process.
- After making changes, select the workflow process in the tree view, and then select **Tools → Validate** to test it.

The BPE displays validation messages that indicate the status of the process:

**Warning indicator (yellow dot)** – Indicates that the process action is valid, but that the syntax style is not optimal.

**Error indicator (red dot)** – Indicates that the process will not run correctly. You must correct the process action.

Click an indicator to display the action it refers to. After making changes, click **Re-validate**. This re-tests the process to make sure the error is corrected and checks for additional errors.

# Standard Workflows

Identity Manager ships with standard workflows that are mapped to commonly used processes. See *Default Workflow Processes* for a brief introduction to these default workflows. To display and edit a default workflow

1. Open the Business Process Editor (BPE). For information on using the BPE, see *Introduction to the Business Process Editor* in *Identity Manager Deployment Tools*.
2. Select **File > Open Repository Object > Workflow Processes**. The BPE displays the Workflow Processes list, which contains the standard workflow processes and any custom workflows in your deployment.
3. Double-click on the name of the workflow you want to display or edit.

You can view process mappings by selecting **Configure > Form and Process Mappings**.

# Synchronize User Password Workflow

Used by the PasswordSync application to synchronize user passwords. Called by the JMS Listener adapter, this workflow takes each request sent by the adapter and checks out, then subsequently checks in, the Change User Password view. After the workflow has completed check-in, the workflow by default launches a confirmation email to the user, specifying whether the check-in passed or failed.

## Variables

The Synchronize User Password workflow contains the following variables.

### IDMAccountId

Specifies the Identity Manager Account ID that was resolved by the message mapper. Null indicates that the user was not found in Identity Manager.

### password

Identifies the password that should be synchronized to all resources.

### sourceResource

Identifies the name of the resource where the resources account was found in Identity Manager.

### userEmail

Specifies the email address of the user. This value is derived either from Active Directory. If null, this is the email address stored in Identity Manager.

### PasswordSyncThreshold

Specifies the configured number of seconds to allow before triggering synchronization. This setting ensures that passwords updated from the Identity Manager User Interface and sent to source resources will not trigger a recursive synchronization of the user's password.

### lastSourcePasswordUpdate

Indicates the last time (in milliseconds) that Identity Manager updated the source resources password. This setting ensures that passwords reset from the Identity Manager User Interface and sent to a resource will not again trigger a synchronization of the user's password.

### lastSourcePasswordSecs

Indicates how many seconds ago the password was updated.

### PasswordSyncResourceExcludeList

Lists the resource names that should always be excluded from synchronization. This setting can be configured by editing the System Configuration object attribute `PasswordSyncResourceExcludeList`.

### resourceAccountId

Specifies the native `accountId` coming from the source resource.

## Activities

### checkForIDMAccountID

Checks for a null `IDMAccountId`. If null, transitions to an empty activity called `noIDMAccountForResourceAccount`. If a non-null account exists, the workflow then transitions to `checkLastUpdateThreshold`.

### noIDMAccountForResourceAccount

Does nothing by default, but is called only if the MessageMapper did not resolve the incoming resource account ID or GUID into an Identity Manager user name.

### checkLastUpdateThreshold

Confirms that the `lastUpdateSecs` variable is less then the `PasswordSyncThreshold`. This activity is used to prevent recursive password resets when a password is changed from the Identity Manager product. For example, a user changes his resource passwords, which include Active Directory. Then, the password notifier tells Identity Manager that the password had just been changed. Without this check, Identity Manager would perform an extra password change. If the `passwordThreshold` had been exceeded we would continue to the checkoutView activity.

### CheckoutView

Checks out the `ChangeUserPassword` for the given Identity Manager account. If there is a successful check out, the workflow transitions to `SetPasswordView`. Otherwise, it transitions to `NotifyFailure`.

### SetPasswordView

Iterates over the resource accounts and selects all resources that are not either the source resource or in the exclude list. All other resources are selected, and the `expiredPassword` flag is set to false to avoid the password having to be changed immediately afterwards. This activity always transitions to `CheckTargets`.

### CheckTargets

Checks to assure we got at least one other target to synchronize. If Identity Manager does not find any other resource accounts, the workflow transitions to `UnlockView`.

### UnlockView

Unlocks the currently checked out view and transitions to end.

### CheckinView

Checks the view in, if there is a failure transitions to `NotifyFailure`. Otherwise, it transitions to `Notify`.

### Notify

Notifies the end user of a successful synchronization and uses the PasswordSyncNotice email template. This activity always transitions to end.

### NotifyFailure

Notifies the end user of a failure synchronization and uses the PasswordSyncNotice email template. This activity always transitions to end.

# Customizing a Process

You can change one or more of the Identity Manager processes to eliminate steps, include new steps, or customize existing steps. Each step in the process is represented by an activity.

The Workflow Toolbox facilitates workflow changes by providing pre-defined activities you can use when editing or creating a workflow.

## About the Workflow Toolbox

Use the toolbox to quickly drag-and-drop workflow activities into the BPE diagram view. Toolbox activities are divided into multiple categories. Click to display activities in each area:

- Workflow Services
- Approvals
- Users
- Workflow Task

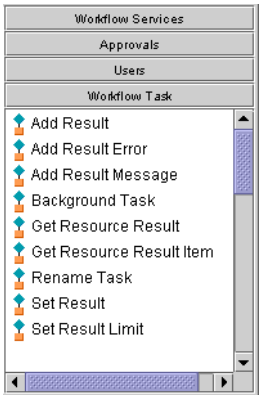



Figure 20. Workflow Toolbox: Workflow Tasks Area

To open the toolbox, right-click in the diagram view and select the toolbox option.

## Default Activities

By category, these default activities are available.

# Default Workflow Services

| Activity | Description |
|---|---|
| Add Deferred Task | Adds deferred task scanner information to an object. |
| Audit Object | Creates audit log records. |
| Authorize Object | Tests authorization for a subject on an object in the repository. |
| Checkin Object | Commits changes to an object. |
| Checkin View | Commits an updated view. |
| Checkout Object | Locks and retrieves a repository object for editing. |
| Checkout View | Gets an updateable view. |
| Create Resource Object | Creates a resource object. |
| Create View | Initializes a new view. |
| Delete Resource Object | Deletes a resource object. |
| Deprovision Primitive | Deprovisions resource accounts. |
| Disable Primitive | Disables resource accounts. |
| Disable User | Disables an Identity Manager user account, resource accounts, or both. |
| Email Notification | Sends email notification of an action. |
| Enable Primitive | Enables resource accounts. |
| Enable User | Enables an Identity Manager user account, resource accounts, or both. |
| Get Object | Retrieves a repository object. |
| Get Property | Retrieves a property. |
| Get View | Gets a read-only view. |
| Manage Resource | Launched when a resource is created from the Identity Manager interface. Saves the new resource in the repository. |
| Manage Role | Launched when a role is created from the Identity Manager interface. Saves the new role in the repository. |

| Activity | Description |
|---|---|
| Query Object Names | Searches for objects with matching attributes. |
| Query Objects | Searches for objects with matching attributes. |
| Refresh View | Refreshes a view that was previously checked out. |
| Remove Deferred Task | Removes deferred task scanner information from an object. |
| Remove Property | Removes an extended property on an object. |
| Reprovision Primitive | Reprovisions resource accounts. |
| Set Property | Adds an extended property to an object. |
| Unlock Object | Unlocks an object that was previously checked out. |
| Unlock View | Unlocks a view that was previously checked out. |
| Update Resource Object | Modifies an object managed by a resource. |

## Approvals

| Activity | Description |
|---|---|
| Approval | Performs the fundamental single approver process. |
| Approval Evaluator | Recursively evaluates an Approval Definition Object to implement a complex approval process.<br>Allows the form and template to be used to be passed in, but those can be overridden if specified in the set. |
| Identity Manager Approval | Performs the default Identity Manager approval process for assigned organizations, roles, and resources. Uses the Approval Evaluator process. |

| Activity | Description |
|---|---|
| Multi Approval | Distributes approvals among multiple approvers. Users the Approval process for each approver. |
| Notification Evaluator | Recursively evaluates an Approval Definition Object to implement a complex notification process. The structure is expected to be the same as that defined for Approval Evaluator. In the standard workflow, approval definitions and notification definitions are maintained in the same structure. This is not required for a customized workflow. |
| Provisioning Notification | Standard process for notifying administrators after a provisioning operation has completed. |

## Users

| Activity | Description |
|---|---|
| DeProvision | Performs the standard steps to deprovision an existing Identity Manager user, with granular control over resource account deletion, Identity Manager user deletion, unlinking, and de-assignment. Individual resource operations are re-tried until successful. |
| Provision | Performs the standard steps to create a new Identity Manager user and provision resource accounts. Individual resource operations are re-tried until successful. |
| Set Password | Changes the password of the Identity Manager account and resource accounts. |
| Update User Object | Checks out a WSUser object, applies a set of changes, and checks in the object. |
| Update User View | Checks out the user view, applies a set of supplied updates, and checks in the user view. |
| Update View | Applies a collection of changes to any view. |

# Workflow Task

| Activity | Description |
|---|---|
| Add Result | Adds a named data item to the task result. |
| Add Result Error | Adds an error message to the task result. |
| Add Result Message | Adds an informational message to the task result. |
| Background Task | Forces the workflow into the background if it was launched from the Identity Manager Administrator interface. |
| Get Resource Result | Retrieves the result object returned by a resource adapter on the last provisioning operation. |
| Get Resource Result Item | Retrieves one result item from the result object returned by a resource adapter on the last provisioning operation. |
| Rename Task | Renames the task instance in the repository. |
| Set Result | Adds an entry to the task entrance result. This will appear in the workflow summary report. |
| Set Result Limit | Sets the number of seconds the task instance should be retained in the repository when it finishes. A non-negative value indicates that the task instance will be kept for this many seconds after the task has completed.<br>A negative value indicates that the task instance will never be removed automatically. However, you can remove it manually. |

## Default Rename Task

```
<Activity name='Rename'>
   <Action>
      <expression>
         <block>
         <!-- rename in the repository -->
            <invoke name='renameObject'>
            <invoke name='getLighthouseContext'>
               <ref>WF_CONTEXT</ref>
            </invoke>
            <invoke class='com.waveset.object.Type'
                        name='findType'>
               <s>TaskInstance</s>
            </invoke>
            <invoke name='getId'>
            <invoke name='getTask'>
               <ref>WF_CONTEXT</ref>
            </invoke>
            </invoke>
               <ref>name</ref>
               <map>
               <s>user</s> <ref>WF_CASE_OWNER</ref>
               </map>
            </invoke>
             <!-- rename in memory -->
              <invoke name='setName'>
               <invoke name='getTask'>
                  <ref>WF_CONTEXT</ref>
               </invoke>
                  <ref>name</ref>
               </invoke>
            </block>
         </expression>
      </Action>
   </Activity>
```

This method first renames the object in the repository. The method renames the task in memory.

### Using the Default Rename Task

To use the default rename task without customization, include the following action in your workflow:

```
<Action process='Rename Task'>
   <Argument name='name' value='New Task Name'/>
</Action>
```

## Creating an Activity from the Toolbox

To create an activity by using the Workflow Toolbox:

1. Click and hold the activity you want to create from one of the four Toolbox areas (Workflow Services, Approvals, Users, Workflow Task).

**Note** The activity does not use highlighting to indicate that it is currently selected.

2. Drag the mouse cursor into the Workflow diagram view. The activity appears in the view.

**Tip** Any activities you create after the first one are numbered. Renumber similar activities before creating more than two.

# Tracking Workflow Progress

The designated owner of a task can always check on the status of a Workflow task. The owner is usually the person that initiated the task, but ownership can be redefined. Because tasks are objects in the repository, they will also be visible to anyone else with sufficient permissions.

Workflow status is typically represented in the Task List State column by the strings **`executing`**, **`pending`**, **`creating`**, and **`suspended`**. You can add additional, more informative strings summarizing workflow status to this column display.

Implement this feature by adding one of two possible expressions to the WFProcess file:

```
<WFProcess name='queryRoleTask' maxSteps='0'>
   <Status>
     <s>Customized Status</s>
   </Status>
     <Activity id='0' name='start'>
       <Transition to='GetReferencingRoles'/>
     </Activity>
     <Activity name='GetReferencingRoles'>
       <Action id='0'>
     <expression>
```

<Status> can be any XPRESS statement that results in a string. For example,

```
<Status>
   <s>custom string</s>
</Status>
```

or

```
<Status>
   <block>
     <s>not appearing</s>
     <s>custom string</s>
   </block>
</Status>
```

The results of this expression, if any, are displayed in the Status column when a result is pending (for example, **pending (*custom status*)**).

# Enabling Workflow Auditing

*Workflow auditing* is similar to regular auditing, except that workflow auditing stores additional information to enable time computations. Regular auditing reports that an event occurred, but does not indicate *when* the event started and ended. See *Identity Manager Audit Logging* for more information about Identity Manager auditing.

Workflow auditing is enabled by adding the `audit` attribute to a workflow or one or more of its Activities. Once the attribute is in place, activate workflow auditing by selecting the **Audit entire workflow** checkbox in the appropriate task template of the Administrator interface. See the chapter titled “Task Templates” in *Identity Manager Administration* for procedural information about turning on auditing in a task template.

## Overview

Workflow auditing operations store predefined attribute names and their values per audit event. You can enable auditing within a workflow by adding the `audit=”true”` attribute to the `WFProcess` element or to one or more `Activity` elements. Setting the attribute at the `WFProcess` level causes the entire workflow to be audited, while adding the attribute to individual Activity elements causes only certain activities to be audited. If the audit attribute is not set, then auditing is disabled. In addition, auditing must be enabled from within task template that calls the workflow.

The following example enables auditing for the Notify action:

```
<Activity name='Notify' audit='true'>
```

## What Information Is Stored and Where Is It?

By default, workflow auditing collects most of the information stored by a regular audit event, including the following attributes:

- **WORKFLOW**: Name of the workflow being executed
- **PROCESS**: Name of the current process being executed
- **INSTANCEID**: Unique instance ID of the workflow being executed
- **ACTIVITY**: Activity in which the event is being logged
- **MATCH**: Unique identifier within a workflow instance
- **ORGANIZATION**: The name of the user's organization

These attributes are stored in the `logattr` table and are derived from `auditableAttributesList`.

Identity Manager also checks whether the `workflowAuditAttrConds` attribute is defined.

You can call certain activities several times within a single instance of a process or a workflow. To match the audit events for a particular activity instance, Identity Manager stores a unique identifier within a workflow instance in the `logattr` table.

To store additional attributes in the `logattr` table for a workflow, you must define a `workflowAuditAttrConds` list `(`assumed to be a list of `GenericObjects)`. If you define an `attrName` attribute within the `workflowAuditAttrConds` list, Identity Manager pulls `attrName` out of the object within the code — first using `attrName` as the key, and then storing the `attrName` value. All keys and values are stored as uppercase values.

# Sample Workflow Customization

The following sample illustrates the end-to-end steps you would follow to customize a a workflow process.

Example: Email Notification – This two-part example shows how to:

- Part 1 – Create an Identity Manager email template
- Part 2 – Customize the workflow process to use the new template and send email notification to a user that welcomes him to the company

**Notes:**

- The process illustrations included in these examples may differ slightly from those you see when you load a process. Discrepancies can result in different component positioning or the selective removal of process actions that are unimportant to the example.
- You can perform many steps from the tree view or the diagram view in the BPE. The example shows primarily how to perform tasks from the tree view.

# Example: Email Notification

**Goal:** Customize the Identity Manager Create User workflow to send a welcome email to an approved user.

## Part I – Create an Identity Manager Email Template

To create a custom email template, use an existing template and modify it. Follow these steps:

### Step 1: Select the Email Template Object from the Repository

1. From the BPE menu bar, select **File → Open Repository Object**.
2. From the selection dialog, select the Account Creation Notification template, and then click **OK**.

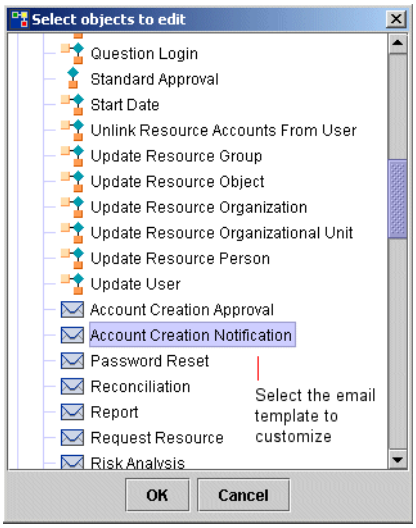



Figure 21. Email Notification Part 1, Step 1 (Select Email Template)

The BPE displays the email template.

### Step 2: Clone the Email Template

Use the copy-and-paste function in the BPE list view to create the new template:

1. Highlight the template, and then right-click and select **Copy**.
2. Right-click again, and then select **Paste**.

A copy of the email template appears in the list view.

**Tip** When pasting, be sure the mouse does not cover an item and that no items are selected (otherwise, the paste action is ignored).

## Step 3: Rename the New Template

1. Double-click the new email template in the list view to open it.
2. Change the name in the Name field to User Creation Notification.

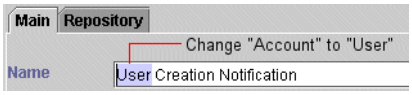



Figure 22. Email Notification Part 1, Step 3 (Rename the New Template)

## Step 4: Modify the Template Contents

Modify the subject and email body of the newly created User Creation Notification template as desired. You can also add a comma-separated list of either Identity Manager accounts or email addresses to the Cc field. Click **OK**.

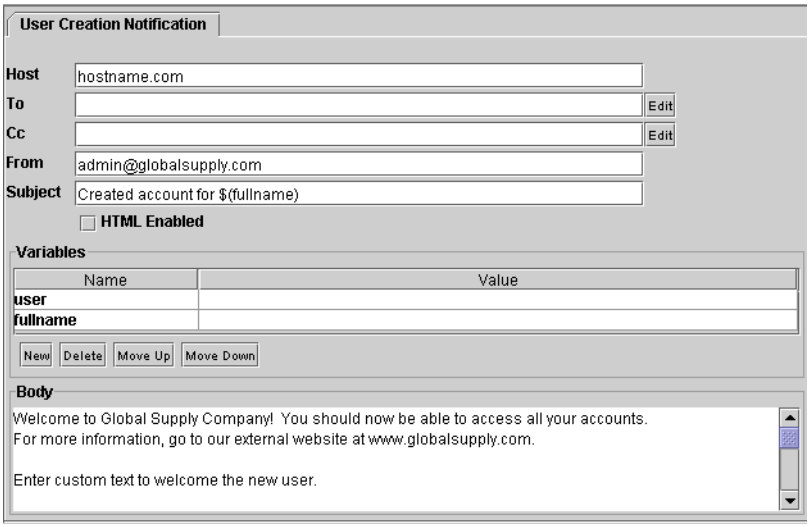



Figure 23. Email Notification Part 1, Step 4
(Customize User Creation Notification Email Template)

## Step 5: Save and Check in the Template

To save and check the template into the repository, select **File → Save in Repository** from the menu bar.

# Part 2 – Customize the Workflow Process

Continue with Part 2, in which you modify the Create User workflow process to use the new email template.

## Step 1: Load the Workflow Process

1. From the BPE, select **File → Open Repository Object**.

   A dialog appears that contains Identity Manager objects you can edit.
2. Select the **Create User** workflow process.
3. Click **OK**.

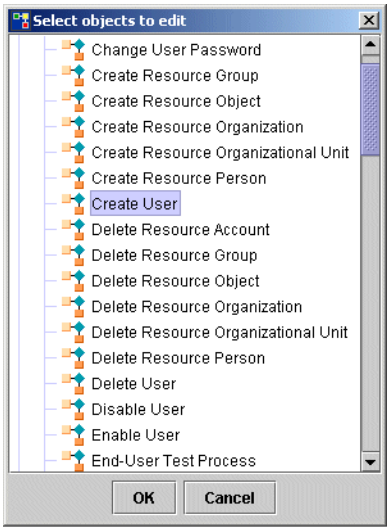



Figure 24. Email Notification Part 2, Step 1 (Load the Workflow Process)

The Create User workflow appears.

## Step 2: Add an Activity Named Email User

1. In the tree view, select the Create User process.
2. Right-click, and then select **New → Activity**.

   The tree view displays a new activity named **activity1** at the bottom of the list of activities.
3. Double-click **activity1** to open the activity dialog.

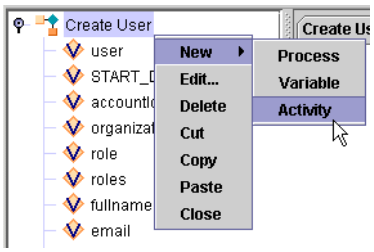



Figure 25. Email Notification Part 2, Step 2 (Create and Name an Activity)

4. In the dialog Name field, change the name to Email User.
5. Click **OK** to save the name and close the dialog.

## Step 3: Create and Modify Transitions

To include the new step in the workflow, you must add and change transitions. In the default Create User workflow, the step that notifies the account requestor that the account was created (**Notify**) transitions directly to **end**. You will delete this transition and create transitions (between **Notify** and **Email User**, and from **Email User** to **end**) to send email to the user before the process ends.

To do this:

1. Right-click **Notify**, and then select Edit.
2. In the Activity dialog Transitions area, select **end**, and then click **Delete** to delete that transition.
3. In the Transitions area, click **New** to add a transition and open the Transition dialog.
4. Select Email User from the list, and then click **OK**.

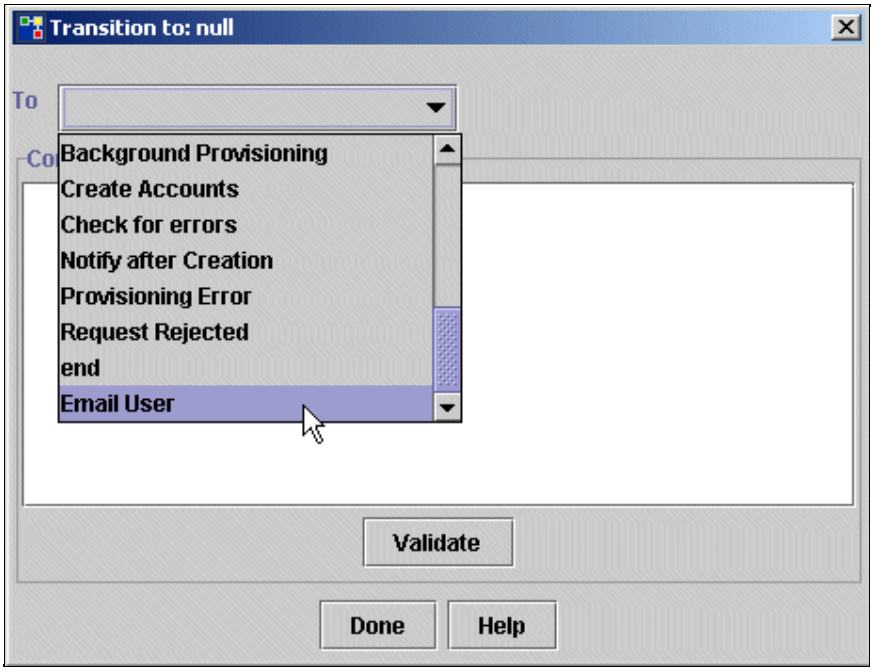



Figure 26. Email Notification Part 2, Step 3 (Create and Modify Transitions)

5. Click **OK** to close the dialog.
6. In the tree view, right-click **Email User**, and then select **New → Transition** to create a transition and open the Transition dialog.
7. Select **end**, and then click **OK**.

## Step 4: Create an Action

Create an action for the new Email User activity that defines the email action and its recipient.

1. In the tree view, right-click **Email User**, and then select **New → Action** to create an action and display the Action dialog.
2. For Type, select the **Applications** option.
3. Enter a name for the new action.
4. For Application, select **email** from the list of options.

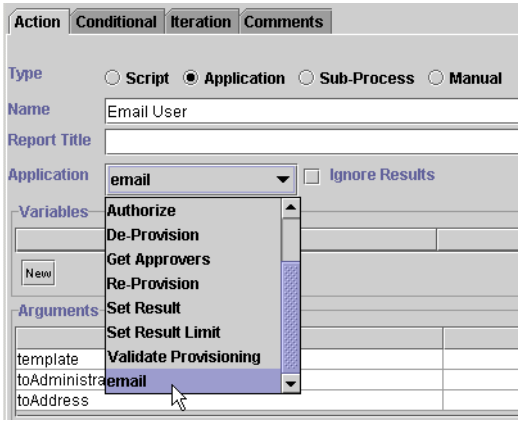



Figure 27. Email Notification Part 2, Step 4 (Create an Action)

The Action dialog displays new selections in the Arguments area.

5. For **template**, enter the new email template name (User Creation Notification).
6. For **toAddress**, enter a variable for the user as `$(user.waveset.email)`.
7. Click **New** to add an argument. Name the argument `accountId` and enter a value for the argument as `$(accountId)`.

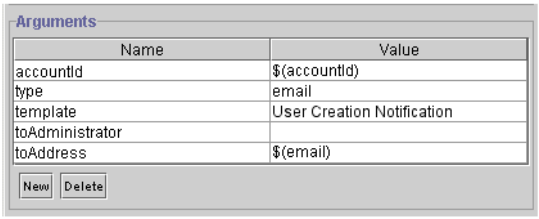



Figure 28. Email Notification Part 2, Step 4 (Create an Action)

8. Click **OK**.

## Step 5: Save and Check in the Modified Workflow

To save the process and check it back into the repository, select **File → Save in Repository** from the menu bar.

### Step 6: Test the New Process

After saving, you can test the new process by using Identity Manager to create a user. For simplicity and speed, do not select approvers or resources for the user. Use your email address (or one that you can access) so that, upon creation, you can verify receipt of the new welcome message.

# Adding Applications

You can register your own Java methods so that they can be accessed from the BPE. To do this:

1. Edit the `idm/config/workflowregistry.xml` file.
2. Add the application definition, in a form similar to this example:

```
<WorkflowApplication name='Increment Counter'
  class='com.waveset.util.RandomGen' op='nextInt'>
   <ArgumentDefinition name='start' value='10'>
      <Comments>Get the next counter</Comments>
   </ArgumentDefinition>
</WorkflowApplication>
```

3. Restart the BPE.

The new application is added to the application menu.

# Integrating Remedy with Workflow

Workflow processes can be integrated with a Remedy server through the Remedy Resource adapter. The Remedy adapter is a gateway-based adapter that runs on a Windows platform.

## Prerequisite

**Confirm that you have successfully configured the Remedy resource adapter by creating a Remedy resource**. To confirm that the adapter has been successfully configured, check **Configure → Managed Resources** through the Identity Manager Administrator interface.

### What Does a Remedy Schema Represent?

Schemas correspond to objects such as trouble tickets and service requests. These schemas are not configured in the adapter through the Identity Manager administrator interface, specifically the Remedy Ticket Integration. They are invoked through the workflow subprocess `Configuration:RemedyTicketServices`.

This section discusses the creation of Remedy tickets through Identity Manager workflow.

## Creating a Remedy Integration Template

To define a Remedy template, select **Configure → Remedy Configuration**. You then select the Remedy schema and resource.

Once the Remedy RA is defined, you define Remedy Integration templates through the Configure menu on the Identity Manager administrator interface. These templates are referenced in workflow and are used for opening and updating Remedy trouble tickets. Although the schema typically varies from customer to customer, the Remedy Integration tool is an excellent mechanism for verifying the existence of a particular schema definition and for retrieving information such as field names, field IDs, and drop-down lists of acceptable values.

### Template Attributes

Remedy Template attributes are described below:

| Template Attribute | Description |
|---|---|
| `Select` | Identifies one or more rows in schema definition for removal. |
| `Field Name` | Specifies the display name in Remedy for schema attribute |
| `Field ID` | Specifies the internal numeric reference to the schema attribute in Remedy. These would be the right-side values of the schema map for the Remedy RA in the case of the user profile. |
| `Field Value` | Identifies either the default value for the field or the workflow variable to associate with this field. If the field is set in workflow, the syntax for the field value is of the form `$(<wf attr>)` so that `assigning user.account.wavesetId` would look like `$(user.account.wavesetId)`. |
| `Choices` | Constrained list of values, if applicable. |

You define and modify the configuration object at the Remedy Integration configuration page in the Identity Manager Administrator interface. The RemedyTemplate works similar to an e-mail template in that it allows replacement of variables at runtime. The typical convention of `$(someAttribute)` is used similar to email templates. However, in the case of Remedy Templates, the system attempts to resolve variables in three different ways, in the following order:

## Creating a Template

To create a template

1. Specify the names of the following attributes:
   - Template
   - schema definition for the tickets that are being managed
   - Remedy resource adapter that is used to handle the ticket requests
2. Select **Discover Required Fields** to retrieve the schema definition.

You can then use the integration template to test the creation of objects of a particular schema by hard-coding the field values in the template for required fields and selecting **Test**.

## Order of Variable Resolution

Integration template variables are resolved in the following order:

1. **Workflow context.** The workflow context is automatically available to the RemedyTicketServices WorkflowApplication. During variable resolution (denoted with the `$()` syntax in the Remedy Integration template), the workflow context variables map are searched. For example, `$(user.accounts[Lighthouse].fullname)` would be valid during the standard provisioning workflow.
2. **Action arguments**. Arguments to the RemedyTicketServices workflow application are then examined for variables that are not resolved from the workflow context.
3. **User-supplied value map.** If a value map is specified as the value of the `remedyTicketValueMap`, this value map is then searched.

Additionally, static values that are specified directly in the Remedy Integration template (*not* denoted with the `$()` syntax) are assimilated directly into all requests.

**Note** One special argument value called `TRAVERSE_RESULT`. If specified, this class replaces the value with a String version of any errors in the current case waveset result. You can configure the RemedyTemplates from the Identity Manager user interface at the Remedy Integration configuration page. For example, specifying `$(TRAVERSE_RESULT)` as the value to the `worklog` field logs the current case errors.

# Populating Remedy Template Fields

There are two mechanisms for populating fields in Remedy templates that are generated as result of a workflow:

- Remedy Workflow process
- RemedyTicketServices workflow

## Using the Remedy Workflow Process to Populate Template Fields

The Remedy Workflow Process subprocess provides access to the RemedyTicketServices Workflow application. As a Configuration object in Identity Manager (identified by `id='#ID#Configuration:RemedyTicketServices`), the subprocess can be called from any point in workflow. This is especially convenient under the following circumstances:

- additional logic that is specific to Remedy is required in constructing trouble ticket requests
- you do not want the logic duplicated across workflows definitions

**Note** When using the Remedy Workflow Process, note that you cannot specify argument values explicitly to the application without modifying this workflow. Only variables named as arguments to the RemedyTicketServices application are defined as inputs to the subprocess. Remedy Template values are passed to the Remedy Workflow Process either through top-level variables declared (and set) in the calling process or through the remedyTicketValueMap passed as an argument to the Remedy Workflow process.

The Remedy Workflow Process process wraps the call to this application to streamline the workflow implementation. The process returns the `ticketId` variable, which can be used later in the workflow, if necessary.

```
<Activity name='openTicket'>
   <Action id='0' name='RemedyTicketServices'>
      </ReportTitle>
         <Variable name='op' value='open'/>
         <Variable name='remedyTemplate' value='OpenTicketTemplate'/>
         <Variable name='ticketId'/>
         <Return from='ticketId' to='ticketId'/>
         <ExternalProcess>
         <ObjectRef type='Configuration'
         id='#ID#Configuration:RemedyTicketServices' name='Remedy
         Workflow Process'/>
         </ExternalProcess>
   </Action>
   <Transition to='end'/>
</Activity>
```

## Using the RemedyTicketServices Workflow to Populate Template Fields

You open and update trouble ticket objects through workflow. The RemedyTicketServices workflow application is provided as the interface for generating requests into Remedy. The following sample provides syntax for invoking this application:

```
<Action id='0'
application='com.waveset.helpdesk.RemedyTicketServices'>
   <Argument name='op' value='update'/>
   <Argument name='remedyTemplate'
      value='RemedyIntegrationTemplateName'/>
   <Argument name='remedyTicketValueMap'>
      <map>
         <s>key1</s>
         <ref>value1</ref>
         <s>key2</s>
         <ref>value2</ref>
      </map>
   </Argument>
   <Argument name='remedyResource' value='RemedyResourceName'/>
   <Argument name='ticketId' value='$(ticketId)'/>
</Action>
```

The RemedyTicketServices workflow also retrieves information out of the workflow context of the calling process to resolve variables identified in the Remedy Integration template.

### Supported Arguments for RemedyTicketServices

The following table describes the options that the RemedyTicketServices application can take.

| Argument | Description |
|---|---|
| `op` | (Required) Specifies the operation for the ticket. Valid setting either **`open`** or **`update`**. |
| `remedyTemplate` | (Required) Specifies the name or ID of the `remedyTemplate` to use during the operation. |
| `ticketId` | (Required during updates) Variable returned after an open ticket request. This value must be stored if you later want to update the ticket. |
| `remedyResource` | (Optional) Identifies the name or ID of the resource to use when interacting with Remedy. The template defines a resource but can be overridden with this value, if necessary. |
| `remedyTicketValueMap` | (Optional) Maps the values to use when resolving $(references) specified in the template field values. |

## Configuring Workflow Properties

The System Configuration object controls workflow configuration properties. The following table lists the most frequently configured properties.

For information on using the Business Process Editor to view or edit the System Configuration object, see *Working with Generic and Configuration Objects* in *Introduction to the Business Process Editor*.

| Attribute Name | Data Type | Default Value |
|---|---|---|
| `consoleTrace` | String | false |
| `fileTrace` | null | |
| `maxSteps` | String | 5000 |
| `resultTrace` | String | false |
| `retainHistory` | String | false |

| Attribute Name | Data Type | Default Value |
|---|---|---|
| traceAllObjects | String | false |
| traceLevel | String | 1 |
| validationLevel | String | CRITICAL |

### consoleTrace

Specifies whether trace messages are emitted to the console. When set to `true`, trace messages are emitted. Default is false.

### fileTrace

Specifies whether trace messages are emitted to a named file. To send trace messages to a specific file, enter the file name.

### maxSteps

Specifies the maximum number of steps allowed in any workflow process or subprocess. Once this level is exceeded, Identity Manager terminates the workflow. This setting is used as a safeguard for detecting when a workflow is stuck in an infinite loop. To override the default setting on a per-workflow basis, set a new value in the Maximum transitions field of the Business Process Editor Process dialog.

The default is 5000 steps.

### resultTrace

Specifies whether trace messages should be retained in the task's result object. When set to true, trace messages accumulate in the task's result object.

### retainHistory

Indicates whether the entire history should be returned after the task has completed. When set to `true`, Identity Manager returns the entire case history. Although it can useful to examine the history when diagnosing process problems, complete results can be large.

### traceAllObjects

Indicates whether generic objects should be included in workflow trace operations.

### traceLevel

Specifies the level of detail included in workflow trace. An unspecified or 0 value generates the most detail. The default is 1.

### validationLevel

Identifies the level of strictness that is applied when validating workflows prior to running them. Errors of this level or greater will result in the workflow not being run. Valid values are CRITICAL, ERROR, WARNING, or NONE, where NONE turns off validation completely.

The default is CRITICAL.

# 2 Workflow Services

This chapter describes each of the Identity Manager workflow services that are available to the Identity Manager implementer who is modifying or creating custom workflows.

Identity Manager contains default workflows to define the process for provisioning and manipulating user accounts. During a customer implementation, you can modify these workflows to reflect the customer's business rules. Workflow allows a customer's business rules for account provisioning to be implemented in Identity Manager.

For information on using workflows, see *Workflow*.

## Workflow Built-in Variables

The workflow engine uses several built-in variables. Most of these variables do not need to be declared in the workflow. Built-in variables can be used to find out the state of the workflow execution. In addition, many variables are set as a side effect of workflow services.

| Name | Description |
|---|---|
| WF_ACTION_ERROR | A built-in variable that will be set to `true` if the previously executed action returned a result containing an error or a thrown exception. |
| WF_ACTION_RESULT | A built-in variable that will be set to the WavesetResult object returned by the previous action. Use this variable when you want to capture the action's WavesetResult and process it without adding it to the global TaskInstance result. It was originally added to support resource retries, where you do not necessarily want to keep adding the resource error messages to the task result on every retry. It is not used often, but can be useful if you ever want to massage the action result before adding it to the task result. |
| WF_ACTION_SUPPRESSED | This built-in variable will be set to `true` if the action was suppressed due to a `<Condition>` expression evaluating to false. |
| WF_ACTION_TIMEOUT | A built-in variable that will be set to `true` if the previously executed action timed out. |

| Name | Description |
|---|---|
| WF_CASE_OWNER | A built-in variable that contains the name of the administrator that launched the workflow task. |
| WF_CASE_RESULT | A built-in variable that contains the WavesetResult of the TaskInstance. This can be used in Actions implemented in XPRESS or JavaScript to get a hold of the result. For Actions implemented as WorkflowApplication classes, they can obtain the result through the WorkflowContext. Since the entire WorkflowContext is exposed through the WF_CONTEXT variable, this is not absolutely necessary, but convenient. |
| WF_CONTEXT | A built-in variable that contains a WorkflowContext object. This can be used in Actions implemented in XPRESS or JavaScript to get a hold of the WorkflowContext. For Actions implemented as WorkflowApplication classes, the context is passed in. |
| WF_SESSION_APPLICATION | A built-in variable that contains the name of the application that was used when the workflow process was launched. The values are `Administrator Interface`, `User Interface`, `Command Line Interface`, `IVR Interface`, and a few others. This is used only when generating audit log records. |
| WF_TRACE | A built-in variable that causes trace message to be accumulated. Unlike the other built-in variables, this is not set by the engine, it must be declared in the workflow and set to `true` if you want to enable trace. |
| WF_WORK_ITEMS | Not a workflow variable. It is the name of an item in the WavesetResult that we maintain for the workflow wizard feature in the Identity Manager interface. The pages that launch processes and edit work items look here to see if there is another work item for the current user. If there is, the pages transition to the work item edit page automatically. |

# General Session Workflow Services Call Structure

Workflow services are called from workflow actions. The general form of a session workflow service action is:

```
<Action class='com.waveset.session.WorkflowServices'>
   <Condition/>
   <Argument name='op' value=workflowServiceOp/>
   <Argument name=argname1>
      <expression>value1expression</expression>
   </Argument>
   <Argument name=argname2>
      <expression>value2expression</expression>
   </Argument>
                  ...
   <Argument name=argnameN>
      <expression>valueNexpression</expression>
   </Argument>
</Action>
```

Each of the supported workflow services has a variable number of required and optional arguments.

## Supported Session Workflow Services

Identity Manager currently supports the following session workflow services. The *op* argument to the session workflow services call must be one of these values.

- addDeferredTask
- audit
- authorize
- checkinObject
- checkinView
- checkoutObject
- checkoutView
- checkStringQualityPolicy
- createView
- disableUser
- enableUser
- findUser
- getObject

- `getProperty`
- `getResource`
- `getView`
- `getViewForm`
- `listResourceObjects`
- `queryObjectNames`
- `queryObjects`
- `queryReferencingRoles`
- `refreshView`
- `removeDeferredTask`
- `removeProperty`
- `setProperty`
- `unlockObject`
- `unlockView`

If an *`op`* argument is given that is not on the above list, the workflow services return:

`'Unknown WorkflowServices op'`

and the workflow context variable `WF_ACTION_ERROR` will be non-null.

# addDeferredTask Session Workflow Service

Used to set up the properties that are recognized by the Deferred Task Scanner task. Deferred Task Scanner typically iterates over WSUser objects looking for a property.

When using this method to set a deferred task on a user, you can set arbitrary values using the *taskDefinition* argument. This argument must be an object that contains the arbitrary values. These values will be passed into the workflow that is run when the deferred task is triggered.

This method supports the ability to add multiple versions of the same task definition to a user. See the description of the *taskDefinition* argument for more information.

**Note** The built-in arguments, such as *date* and *description*, are already made available to the called workflow.

## Example

```
<Activity name='setDefTask'>
     <Action application='com.waveset.session.WorkflowServices'>
       <Argument name='op' value='addDeferredTask'/>
       <Argument name='name' value='$(user)'/>
       <Argument name='task' value='$(taskType)'/>
       <Argument name='date' value='$(date)'/>
       <Argument name='description' value='$(taskDescription)'/>
       <Argument name='taskDefinition'>
          <Object>
              <Attribute name='Arg1' value='value1'/>
              <Attribute name='Arg2' value='value2'/>
              <Attribute name='Arg3' value='value3'/>
          </Object>
       </Argument>
     </Action>
     <Transition to='end'/>
</Activity>
```

## Arguments

| Name | Required | Valid Values | Description |
|---|---|---|---|
| *type* | no | list of types | Indicates the type of object that the deferred task will be added to. If not supplied, the type is defaulted to *user.* |
| *name* | yes | | Specifies the name of the object to which the deferred task will be added. |
| *task* | no | | Identifies the name of the `TaskDefinition` to run. If the taskDefinition argument is a string, then this argument is the name of the GenericObject that will be placed on the property list. |
| *taskDefinition* | no | | Specifies the task definition. Can be either a string or Generic Object.<br>• String -- If this argument takes the form of a string, then it defines either the Task Definition or Task Template name, and the *task* argument is the name of the GenericObject that will be placed on the property list.<br>• Generic Object -- If this argument takes the form of a Generic Object, the *taskDefinition* argument is a preconstructed property listObject, which can in turn contain a taskDefinition attribute. |
| *instanceName* | no | | Identifies the name of the `TaskInstance` to create. |
| *date* | no | Can be either a `java.util.Date` object or a string in the usual format | Specifies the date on which you want the deferred task to run. |
| *organization* | no | | Indicates the organization in which to put the `TaskInstance`. |

| Name | Required | Valid Values | Description |
|---|---|---|---|
| *owner* | no | | Specifies the name of the effective owner of the `TaskInstance` (used for later management). |
| *description* | no | | Specifies optional descriptive text to include with the `TaskInstance` for the task management page. |
| *executeOnce* | no | | When set to `true`, indicates that the deferred task will be successfully executed only once.<br>If an error occurs during task execution, the task will continue to be executable by the deferred task scanner until the task completes successfully one time or is removed manually through a call to the `removeDeferredTask` method. |

# audit Session Workflow Service

Requests Identity Manager to record an audit event using Identity Manager's audit services. The audit will be logged under the workflow administrator that launched the workflow (`WF_CASE_OWNER`).

## Arguments

| Name | Required | Valid Values | Description |
|---|---|---|---|
| *op* | yes | `audit` | |
| *type* | yes | see *Type Names* | The Identity Manager type of the object that is being audited. |
| *logResultErrors* | no | true/false | If `false`, this method logs the single error indicated by the other parameters. If `true`, it creates a log record for each error message in the task results. The task result is obtained from the workflow context. This is a convenient way to log all the errors accumulated during workflow execution. |
| *name* | no | | Identifies the name of the object that is being audited. |
| *status* | no | success<br>failure | Indicates the audit status. If no status is supplied, the status is assumed to be failure. |
| *reason* | no | | Currently hard-coded to `DATABASE_ACCESS_FAILED` (any value you use will be ignored). |
| *action* | yes | see *Action Names* | Indicates the Identity Manager action to audit. |
| *resource* | no | | Specifies the resource that is being audited. |
| *accountId* | no | | Identifies the `accountId` that is being audited. |
| *error* | no | | Indicates the error that is being audited. |

| Name | Required | Valid Values | Description |
|---|---|---|---|
| *parameters* | no | | This is expected to be a `java.util.Map` object that contains name/value pairs. These pairs are specific to each event. For example, the parameter named *`RoleName`* is typically set for log records that are related to users. |
| *attributes* | no | | This is expected to be a `java.util.Map` object that contains name/value pairs. This is formatted in the log record by the audit system in the following format: `name1=value1;; name2=value2;;`<br>It is typically used to log changes to specific account attributes. |
| *organizations* | no | | Specifies a list of the organizations (also known as `ObjectGroups`) associated with the object being audited. |

## Return Values and Side Effects

An audit log is entered with the supplied information.

# authorize Session Workflow Service

Requests that Identity Manager checks if the subject of the workflow is authorized with the given right for the given action on the given object. The subject of the workflow is the administrator that the workflow is running under. This will be the value of the `WF_CASE_OWNER` built-in variable.

## Arguments

| Name | Required | Valid Values | Description |
|---|---|---|---|
| *op* | yes | authorize | |
| *type* | yes | see *Type Names* | Specifies the type of the object that is being authorized. |
| *name* | yes | | Indicates the name of the object that is being authorized. |
| *right* | yes | see *Right Names* | Indicates the right to authorize. |
| *action* | yes | see *Action Names* | Specifies the action to authorize. |

## Return Values and Side Effects

If the WF_CASE_OWNER is not authorized to for the given right or action on the given object, WF_ACTION_ERROR will be `true`. The WavesetResult will contain the error.

# checkStringQualityPolicy Session Workflow Service

Checks string quality against a specified policy.

## Arguments

| Name | Required | Valid Values | Description |
|---|---|---|---|
| *policy* | yes | | Identifies the policy (String) |
| *map* | no | | Provides a map of the data that the string must not contain (Map) |
| *value* | yes | | Specifies the value of the string to be checked. (String) |
| *pwdhistory* | no | | Lists user's password history. |
| *owner* | yes | | Identifies the user whose string value is being checked. |
| *object* | | | Specifies the string object |

## Return Values and Side Effects

`true` -- Indicates that the string passes the policy test.

The method returns an error message if the string fails the policy test.

# checkinObject Session Workflow Service

Persists the current memory representation of the given object. Identity Manager makes sure that the administrator that launched the workflow is authorized to check in the object.

**Note** Using views is more convenient than using objects. Try using views to modify objects. For information on Views, see *Identity Manager Technical Deployment* guide.

## Arguments

| Name | Required | Valid Values | Description |
|---|---|---|---|
| *op* | yes | `checkinObject` | |
| *object* | no | | Indicates the object that is being checked in. If no object is supplied, the workflow service has no effect. |
| *lockedBy* | no | | Specifies the alternate lock name that was used when locking the object. |

## Return Values and Side Effects

The object specified is checked in. If any error occurred, `WF_ACTION_ERROR` will be `true`, and the WavesetResult will contain the error.

# checkinView Session Workflow Service

Checks in a view.

## Arguments

| Name | Required | Valid Values | Description |
|---|---|---|---|
| *op* | yes | `checkinView` | |
| *view* | no | | Identifies the view to be checked in. If no view is specified, the `checkinView` workflow service has no effect. |

## Return Values and Side Effects

The specified view is checked in to the Identity Manager repository. If any error occurred, `WF_ACTION_ERROR` will be `true`, and the WavesetResult will contain the error.

# checkoutObject Session Workflow Service

Obtains and locks a persistent Identity Manager object. Identity Manager makes sure that the administrator that launched the workflow is authorized to check out the object.

**Note** Using views is more convenient than using objects. Try using views to modify objects. For information on views, see *Views*.

## Arguments

| Name | Required | Valid Values | Description |
|---|---|---|---|
| *op* | yes | `checkoutObject` | |
| *type* | yes | | Indicates the type of the object that is being checked out. |
| *name* | yes | | Specifies the name of the object that is being checked out. |
| *lockedBy* | No | | Gives an alternate name to use to record the lock. |
| *lockWait* | No | | Indicates the number of seconds to wait if there is contention on a lock. |

## Return Values and Side Effects

If the checkout is successful, the object checked out will be placed in the variable *object*. If the lock timed out, `WF_ACTION_TIMEOUT` will be set to true for the checkout action. If any other error occurred (including administrator not authorized to check out the object), `WF_ACTION_ERROR` will be `true`, and the WavesetResult will contain the error.

# checkoutView Session Workflow Service

Fetches and locks an Identity Manager view.

## Arguments

| Name | Required | Valid Values | Description |
|---|---|---|---|
| *op* | yes | `checkoutView` | |
| *viewid* | no | | Identifies the ID of the view to checkout. If this argument is not specified, both the type and ID must be specified. Type and ID may be easier to use from workflows. |
| *name* | yes | | Specifies the name of the object that is being checked out. |
| *type* | no | | Identifies the type for the view. |
| *id* | No | | The ID of the object being manipulated in the view. For example, if you are creating a view to manipulate an account object, this would be the `accountId`. |

| Name | Required | Valid Values | Description |
|---|---|---|---|
| raiseViewErrors | no | true or false | Specifies that display.errors is promoted to WF_ACTION_ERROR. When set to true, errors in the display.errors attribute of the view are automatically extracted and returned to the workflow engine as errors. The WF_ACTION_ERROR workflow variable is set. |
| subject | no | | The Identity Manager administrator to check out the view under. If this is not supplied, the administrator that launched the workflow will be used (WF_CASE_OWNER) |
| options | no | | View-specific options. The values you can pass are specific to the view being used. The most common is the user view. Options can be found in session.UserViewConstants. The simpler views should declare their option constants in the *Viewer.java file. Probably the second most common view used from workflow is ProcessViewer, followed by PasswordViewer, DisableViewer, EnableViewer, and RenameViewer. These have comparatively few options. |

## Return Values and Side Effects

If the checkout is successful, the view checked out will be placed in the variable: *view*. If the lock timed out, WF_ACTION_TIMEOUT will be set to true for the checkout action. If any other error occurred (including administrator not authorized to check out the view), WF_ACTION_ERROR will be true, and the WavesetResult will contain the error.

# createView Session Workflow Service

Creates an empty object of the type given with the appropriate form.

## Arguments

| Name | Required | Valid Values | Description |
|---|---|---|---|
| *op* | yes | `createView` | |
| *viewid* | no | | Indicates the ID of the view to be created. If this argument is not specified, both the type and ID must be specified. Type and ID can be easier to use from workflows. |
| *type* | no | | Specifies the type for the view |
| *id* | no | | Identifies the ID of the object being manipulated in the view. For example, if you are creating a view to manipulate an account object, this would be the `accountId`. |
| *options* | no | | Specifies view-specific options. The values you can pass are specific to the view being used. The most common is the user view. Options can be found in `session.UserViewConstants`. The simpler views should declare their option constants in the `Viewer.java` file. Probably the second most common view used from workflow is ProcessViewer, followed by PasswordViewer, DisableViewer, EnableViewer, and RenameViewer. These have comparatively few options. |

## Return Values and Side Effects

If the create is successful, the view will be placed in the variable *view*. If any error occurred, `WF_ACTION_ERROR` will be `true`, and the WavesetResult will contain the error.

# disableUser Session Workflow Service

Disables one or more of a user's resource accounts.

There are three ways to disable an account:

- Set `com.waveset.provision.WorkflowServices op=disable`. See the disable Provision Workflow Service in this chapter. This is the lowest level disable method. It is implemented directly by the provisioning engine. No workflow, authorization, or auditing is involved.
- Set `com.waveset.session.WorkflowServices op=disableUser`.
- Set `com.waveset.session.WorkflowServices op=checkoutView` with arguments:

```
type=DisableViewer
id=<user account id>
```

The second and third option are essentially the same. The second option checks out and checks in the Disable view. This is the equivalent of disabling a user from the Identity Manager interface. It launches the Disable User workflow as a side effect.

Use the provision.`WorkflowServices` method if you are implementing a customized Disable User process. Use one of the other approaches if you are implementing some other process and want to use the standard Disable User process to accomplish the disable.

## Arguments

| Name | Required | Valid Values | Description |
|---|---|---|---|
| *op* | yes | `disableUser` | |
| *doWaveset* | no | true/false | Specifies the ID of the view to be created. If this argument is not specified both the type and ID must be specified. Type and ID can be easier to use from workflows. |
| *services* | no | | Identifies a list of resources to disable. If this argument is not supplied, all of the user's resource accounts will be disabled. |

## Return Values and Side Effects

The specified resource accounts are disabled. If any error occurred, `WF_ACTION_ERROR` will be `true`, and the WavesetResult will contain the error.

# enableUser Session Workflow Service

Enables one or more of a user's resource accounts.

## Arguments

| Name | Required | Valid Values | Description |
|---|---|---|---|
| *op* | yes | `enableUser` | |
| accountId | yes | | Identifies the Identity Manager user to enable accounts for. |
| *doWaveset* | no | true/false | If `true`, the Identity Manager account is enabled for this user. If not supplied, it defaults to `true`, and the account is enabled. |
| *services* | no | | Identifies a list of resources to enable. If this argument is not supplied, all of the user's resource accounts will be enabled. |

## Return Values and Side Effects

The specified resource accounts are enabled. If any error occurred, `WF_ACTION_ERROR` will be `true`, and the WavesetResult will contain the error.

# findUser Session Workflow Service

Finds a user in the Identity Manager repository. This functionality is older than the more flexible query method workflow services. Consider using the `query` method workflow services.

## Arguments

| Name | Required | Valid Values | Description |
|---|---|---|---|
| *op* | yes | `findUser` | |
| attribute Name | yes | | Identifies the name of the Identity Manager queryable attribute to search for the user on. |
| *attribute Value* | yes | true/false | Specifies the value of the queryable attribute to search for the user on. |

## Return Values and Side Effects

The `accountId` of the first user found that contains the given value for the given attribute will be placed in the variable: *`accountId`*. If any error occurred, `WF_ACTION_ERROR` will be `true`, and the WavesetResult will contain the error.

# getObject Session Workflow Service

Returns the Identity Manager object of the given name and type. The administrator that launched the workflow (`WF_CASE_OWNER`) will be check to see if it is authorized to get the object.

**Note** Using views is more convenient than using objects. Try using views to view objects. See *Views* for more information.

## Arguments

| Name | Required | Valid Values | Description |
|---|---|---|---|
| *op* | yes | `getObject` | |
| *type* | yes | see *Type Names* | Identifies the type of the object that is being fetched from Identity Manager. |
| *name* | yes | | Specifies the name of the object that is being fetched from Identity Manager. |

## Return Values and Side Effects

The Identity Manager object with the given type and name will be placed in the variable: *`object`*. If any error occurred, `WF_ACTION_ERROR` will be `true`, and the WavesetResult will contain the error.

# getProperty Session Workflow Service

Obtains a property value from an Identity Manager object. Identity Manager makes sure that the administrator that launched the workflow is authorized to view the object.

## Arguments

| Name | Required | Valid Values | Description |
|---|---|---|---|
| *name* | yes | `getObject` | Identifies the name of the object whose property is to be retrieved. |
| *propertyName* | no | see *Type Names* | Identifies the name of the property to obtain from the object. If no *propertyName* is given, the service has no effect or return |

## Return Values and Side Effects

If the `propertyName` is found on the object, the property's value will be placed in the variable *propertyValue*. If any error occurs (including administrator not being authorized to view the object), WF_ACTION_ERROR will be `true`, and the WavesetResult will contain the error.

# getResourceObject Session Workflow Service

Retrieves a resource object specified by type and ID from the specified resource.

## Arguments

| Name | Required | Valid Values | Description |
|---|---|---|---|
| `resourceId` | yes | | Specifies a valid resource object. |
| `objectType` | no | see *Type Names* | Specifies a valid object type defined in the resource's `<ObjectType>` section. |
| `objectId` | | | Identifies a valid fully qualified object identifier on this resource (for example, `dn`). |
| `options` | | | Controls the behavior of the request.<br>Valid keys include `searchAttrsToGet`, which specifies a list (List) of `objectType`-specific attribute names to get. |

## Return Values and Side Effects

Returns the object (specified by `objectType` and `objectId`) from the specified resource.

# getView Session Workflow Service

Fetches an Identity Manager view. The administrator that launched the workflow (`WF_CASE_OWNER`) will be check to see if it is authorized to get the view.

## Arguments

| Name | Required | Valid Values | Description |
|---|---|---|---|
| *op* | yes | `getView` | |
| viewId | no | | Identifies the ID of the view to fetch. If this argument is not specified, both the type and ID must be specified. Type and ID may be easier to use from workflows. |
| *type* | no | | Indicates the type for the view. |
| *id* | no | | Specifies the ID of the object being fetched in the view. For example, if you are fetching a view to view an account object, this would be the `accountId`. |
| *raiseViewErrors* | no | `true` or `false` | Specifies that `display.errors` is promoted to WF_ACTION_ERROR. When set to `true`, errors in the `display.errors` attribute of the view are automatically extracted and returned to the workflow engine as errors. The WF_ACTION_ERROR workflow variable is set. |
| *options* | no | | Gives view-specific options. The values you can pass are specific to the view being used. The most common is the user view. These options can be found in `session.UserViewConstants`. The simpler views should declare their option constants in the `*Viewer.java` file. Probably the second most common view used from workflow is ProcessViewer, followed by PasswordViewer, DisableViewer, EnableViewer, and RenameViewer. These have comparitively few options. |

## Return Values and Side Effects

If the fetch is successful, the view will be placed in the variable *view*. If any error occurred, `WF_ACTION_ERROR` will be `true`, and the WavesetResult will contain the error.

# getViewForm Session Workflow Service

Returns the form associated with the given view.

## Arguments

| Name | Required | Valid Values | Description |
|---|---|---|---|
| `form` | no | | Identifies the argument that has been replaced by `formId`. |
| `formId` | no | | Identifies the name or ID of the form to get. |
| *`op`* | yes | `getViewForm` | |
| *`view`* | no | | Indicates the view to fetch the form from. If this argument is not supplied, the `getViewForm` workflow service has no effect. |

## Return Values and Side Effects

If the fetch is successful, the form is placed in the variable *`form`*. If any error occurred, `WF_ACTION_ERROR` will be `true`, and the WavesetResult will contain the error.

# listResourceObjects Session Workflow Service

Returns a list of resource object names of the specified resource object type from the specified list of `resourceIds`.

## Arguments

| Name | Required | Valid Values | Description |
|---|---|---|---|
| `objectType` | yes | | Identifies the name of a valid object class for this specified `resourceId`. If null, the method returns objects of all object types as defined by:<br>• the specified `resId` found within the specified search container<br>• the scope and the specified search filter |
| `resourceId` | no | | Identifies a single `resourceId`. This value is derived from the named resource. If this is null, the method looks for `resourceIdsList` argument. |
| `resourceIdList` | yes | | Lists `resourceIds`. These IDs are derived from the named resources. This is only used if the `resourceId` argument is null (undefined). |
| `options` | no | | Indicates a map of options that control the behavior of the search. |

Options are described in the following table.

| Option | Description |
|---|---|
| searchContext | Specifies the context within which to perform the search. If not specified, the method attempts to get a value from RA_BASE_CONTEXT. If no value is available in RA_BASE_CONTEXT, the method searches from the logical top. |
| searchFilter | (Optional) in LDAP search filter format as specified in RFC 1558, of one or more object <attr name> <condition> <value> tuples either and'ed or or'ed together. If not specified, a filter will be constructed using the specified objectType. |
| searchScope | Specifies whether the search should be done on the current object, only within the context of the specified searchContext, or in all subcontext within the specified searchContext.<br>Valid values include object, oneLevel, or subTree. subTree indicates that the search should be performed on all sub contexts within the specified searchContext. |
| searchTimeLimit | Indicates the time limit in milliseconds that a search should not exceed. |
| searchAttrsToGet | Specifies the list of objectType-specific attribute names to get per object. |
| runAsUser | Specifies the user name this request is to be run as. If not specified, defaults to resource proxy admin user. |
| runAsPassword | Indicates the password of runAsUser. Required to authenticate with resource to run the list request as the specified user. |
| cache | Specifies whether the server caches the returned list of objects. A value of true causes the server to cache the returned list of objects. The default is false. |
| cacheTimeout | Set the number of milliseconds before the cache times out (valid only if cacheList is true). When the cache times out, the object will automatically be retrieved from the resource the next time objects are requested. |
| clearCache | Force the caches to be cleared and the objects to be refetched from the resource the next time they are requested. |

# queryObjectNames Session Workflow Service

Returns a list of names that match the query attributes specified.

## Arguments

| Name | Required | Valid Values | Description |
|---|---|---|---|
| *op* | yes | `queryObjectNames` | |
| *type* | yes | | Identifies the object type for which the search is performed. |
| attributes | no | | Indicates a list or map of queryable attribute name/value pairs to use to query objects. If not supplied, all objects of the given type are returned. |
| single | no | `true`<br>`false` | If true, the query will return the name of the first object that matches the query specified, if false or not supplied, the query will return the names of all objects that match the query specified. |

## Return Values and Side Effects

If single is `false` or unspecified, the list of object names that satisfies the query attributes specified is placed into the variable *`queryResult`*. If single is `true`, the name of the first object that satisfies the query attributes specified is placed into the variable *`queryResult`*, If any error occurred, `WF_ACTION_ERROR` will be `true`, and the WavesetResult will contain the error.

# queryObjects Session Workflow Service

Returns a list of objects that match the query attributes specified.

## Arguments

| Name | Required | Valid Values | Description |
|---|---|---|---|
| *op* | yes | queryObjects | |
| *type* | yes | | Indicates the object type for which the search is performed. |
| attributes | no | | Specifies a list or map of queryable attribute name/value pairs to use to query objects. If not supplied, all objects of the given type are returned. |
| single | no | true<br>false | If true, the query will return the first object that matches the query specified. If false or not supplied, the query will return all objects that match the query specified. |

## Return Values and Side Effects

If single is false or unspecified, the list of objects that satisfies the query attributes specified is placed into the variable *queryResult*. If single is true, the first object that satisfies the query attributes specified is placed into the variable *queryResult*, If any error occurred, WF_ACTION_ERROR will be true, and the WavesetResult will contain the error.

# queryReferencingRoles Session Workflow Service

Returns a list of all roles that directly or indirectly reference the specified role.

## Arguments

| Name | Required | Valid Values | Description |
|---|---|---|---|
| *role* | yes | | Specifies the role for which you want to identify associated roles. |

## Return Values and Side Effects

For a given role passed as *role*, this method returns the result in variable *queryResult*, which lists all of the Role objects in the Identity Manager repository that directly or indirectly reference the specified role.

# refreshView Session Workflow Service

Refreshes an Identity Manager view.

## Arguments

| Name | Required | Valid Values | Description |
|---|---|---|---|
| *op* | yes | `unlockObject` | |
| *view* | no | | Identifies the view that is to be refreshed. If this argument is not specified, the `refreshView` workflow service has no effect. |
| *raiseViewErrors* | no | `true` or `false` | Specifies that `display.errors` is promoted to WF_ACTION_ERROR. When set to `true`, errors in the `display.errors` attribute of the view are automatically extracted and returned to the workflow engine as errors. The WF_ACTION_ERROR workflow variable is set. |

## Return Values and Side Effects

If the refresh is successful, the refreshed view is placed in the variable *view*. If any error occurred, `WF_ACTION_ERROR` will be `true`, and the WavesetResult will contain the error.

# removeDeferredTask Session Workflow Service

Used to remove a deferred task from an Identity Manager object. Identity Manager will ensure that the administrator that launched the workflow is authorized to view the object.

## Arguments

| Name | Required | Valid Values | Description |
|---|---|---|---|
| *type* | no | valid values are the list of types | Identifies the type of the object that the deferred task will be added to. If not supplied, the type is defaulted to *user.* |
| *name* | yes | | Specifies the name of the object that the deferred task will be added to. |
| task | no | | Specifies the name of the TaskDefinition to run. |
| taskDefinition | no | | Indicates the complete task definition object. This argument can be used in lieu of specifying all of the other task arguments below. If both a complete task object and any of the other arguments are provided, the singleton arguments override the value of that argument in the task definition object. |
| instanceName | no | | Identifies the name of the TaskInstance to create. |
| date | no | Can be either a `java.util.Date` object or a string in the usual format | Specifies the date on which you want the deferred task to run. |

| Name | Required | Valid Values | Description |
|---|---|---|---|
| organization | no | | Indicates the organization to put the TaskInstance in. |
| owner | no | | Indicates the name of the effective owner of the TaskInstance (used for later management). |
| description | no | | Specifies optional descriptive text to include with the TaskInstance for the Task Management page. |

# removeProperty Session Workflow Service

Removes an Identity Manager object property. Identity Manager makes sure that the administrator that launched the workflow is authorized to view the object.

## Arguments

| Name | Required | Valid Values | Description |
|---|---|---|---|
| *name* | yes | | Indicates the name of the object whose property is to be set. |
| *propertyName* | no | | Specifies the name of the property to modify in the object. If no propertyName is given, the service has no effect or return. |

## Return Values and Side Effects

The Identity Manager object's property is removed. If any error occurs (including administrator not authorized to view the object), WF_ACTION_ERROR will be `true`, and the WavesetResult will contain the error.

# setProperty Session Workflow Service

Sets the value of an Identity Manager object's property. Identity Manager makes sure that the administrator that launched the workflow is authorized to view the object.

## Arguments

| Name | Required | Valid Values | Description |
|---|---|---|---|
| *name* | yes | `setObject` | Indicates the name of the object whose property is wanted |
| *propertyName* | no | see *Type Names* | Specifies the name of the property to obtain from the object. If no `propertyName` is given, the service has no effect or return |
| *propertyValue* | no | the value to set the property to | If no *propertyValue* is given, the property is set to null |

## Return Values and Side Effects

The Identity Manager object's property is set to the given value. If any error occurs (including administrator not authorized to view the object), WF_ACTION_ERROR will be `true`, and the WavesetResult will contain the error.

# unlockObject Session Workflow Service

Unlocks an Identity Manager object. The administrator that launched the workflow (`WF_CASE_OWNER`) is checked to ensure that it is authorized to modify the object. Views are more convenient than using objects. Try using views to accomplish modifications to objects.

## Arguments

| Name | Required | Valid Values | Description |
|---|---|---|---|
| *op* | yes | `unlockObject` | |
| *type* | yes | | Identifies the type of the object that is being unlocked. |
| name | yes | | Specifies the name of the object that is being unlocked. |
| lockedBy | no | | Indicates the alternate lock name that was used when locking the object. |

## Return Values and Side Effects

If successful, the object specified is unlocked. If any error occurred, `WF_ACTION_ERROR` will be `true`, and the WavesetResult will contain the error.

## unlockView Session Workflow Service

Unlocks an Identity Manager view.

### Arguments

| Name | Required | Valid Values | Description |
|---|---|---|---|
| *op* | yes | `unlockview` | |
| *view* | no | | Indicates the view that is to be unlocked. If this argument is not specified, the `unlockView` workflow service will have no effect. |

### Return Values and Side Effects

If successful, the view specified is unlocked. If any error occurred, `WF_ACTION_ERROR` will be `true`, and the WavesetResult will contain the error.

## Provision Workflow Services

There is also a set of services in `com.waveset.provision.WorkflowServices`, although they are used less often than the methods in `com.waveset.session.WorkflowServices`. These are the low-level primitives for performing account management. They are called by the stock workflows. In a custom workflow, you might want to use these or you might want to use the higher level views with `checkoutView/checkinView`, which will in turn launch the stock workflows.

## General Provision Workflow Services Call Structure

Workflow services are called from workflow actions. The general form of a provision workflow service action is:

```
<Action class='com.waveset.provision.WorkflowServices'>
  <Condition/>
  <Argument name='op' value=workflowServiceOp/>
  <Argument name=argname1>
    <expression>value1expression</expression>
  </Argument>
  <Argument name=argname2>
    <expression>value2expression</expression>
  </Argument>
```

```
                ...
   <Argument name=argnameN>
      <expression>valueNexpression</expression>
   </Argument>
</Action>
```

Each of the supported workflow services will have a variable number of required and optional arguments.

## Supported Provision Workflow Services

Following is the list of provision workflow services that Identity Manager currently supports. The *op* argument to the workflow services call must be one of these values.

- approve
- auditNativeChangeToAccountAttributes
- bulkReprovision
- changeResourceAccountPassword
- checkDeProvision
- cleanupResult
- createResourceObject
- deleteResourceAccount
- deProvision
- disable
- enable
- getApprovals
- getApprovers
- lockOrUnlock
- notify
- provision
- questionLock
- reject
- reProvision
- runResourceAction
- unlinkResourceAccountsFromUser
- updateResourceObject
- validate

If an *op* argument is given that is not on the preceding list, the workflow services return:

```
'Unknown WorkflowServices op'
```

and the workflow context variable `WF_ACTION_ERROR` will be non-null.

# approveProvision Provision Workflow Service

Records the approval of a user account.

## Arguments

| Name | Required | Valid Values | Description |
|---|---|---|---|
| *op* | yes | `approve` | |
| *user.waveset.accountId* or *accountId* | yes | | Indicates the name of the Identity Manager user to approve. |
| *options* | no | | Indicates the options to specify provisioning characteristics. See the javadocs for `com.waveset.provision.ProvisioningOptions`. |

## Return Values and Side Effects

A WavesetResult object containing the result of the approval.

# auditNativeChangeToAccountAttributes Provision Workflow Service

Reports native changes to one or more auditable attributes of a resource account.

## Arguments

| Name | Required | Valid Values | Description |
|---|---|---|---|
| *op* | yes | `auditNativeChangeToAccountAttributes` | |
| *user.waveset.accountId* or *accountId* | yes | | Indicates the name of the Identity Manager user to audit. |
| *resource* | yes | | Specifies the resource where native changes took place that are being audited. |
| *prevAttributes* | no | Map | Identifies the map of name/value pairs of the attributes before the change. |
| *newAttributes* | no | Map | Identifies the map of name/value pairs of the attributes after the change. |
| *formattedChanges* | no | Map | Specifies the formatted changes to be logged. |

## Return Values and Side Effects

A WavesetResult object containing the result of the audit.

# bulkReprovision Provision Workflow Service

The method executes a set of queries to find all users that match the given conditions. It then iterates over this list and reprovisions the users one at a time.

To use this method to support automatic reprovisioning on every role update, add the following field to the Role form:

```
<Field name='processInputs.reprovision'>
   <Expansion><s>true</s></Expansion>
</Field>
```

## Parameters

| Parameter | Description |
|---|---|
| *role* | Specifies role name |
| *roles* | Lists role names |
| *organization* | Specifies name of organization |
| *organizations* | Lists organization names |
| *conditions* | Provides a map of query options suitable for LighthouseContext.list Objects |
| *options* | Specifies provisioning options (for example, targets and fetches) |
| *maxErrors* | Specifies the maximum number of errors to tolerate before halting processing |

## Return Values

`successes` - Identifies users that were successfully reprovisioned

`failures` - Identifies users that could not be fully reprovisioned

# authenticateUserCredentials Provision Workflow Service

Authenticates the user against the resource using the password.

## Arguments

| Name | Required | Valid Values | Description |
|---|---|---|---|
| *resourceName* | yes | | Specifies the name of the resource to authenticate against. |
| *accountId* | yes | | Specifies the ID of this account, as it appears on the named resource. |
| *accountPassword* | yes | | Specifies the password for this account (an unencrypted string). |

## Return Values

Returns the result of the call to authenticate. Sets a variable in the WFContext, passwordIsValid, to true on success or anything else on failure.

# changeResourceAccountPassword Provision Workflow Service

Changes the password for one or more resource accounts.

## Arguments

| Name | Required | Valid Values | Description |
|---|---|---|---|
| *op* | yes | `changeResourceAccountPassword` | |
| *resources* | yes | | Specifies the list of resources for which to change the password. |
| *options* | no | Map | Specifies options to specify provisioning characteristics. See the javadocs for `com.waveset.provision.ProvisioningOptions`. |

## Return Values and Side Effects

A WavesetResult object containing the result of the password change.

# cleanupResult Provision Workflow Service

Removes null ResultErrors from the task result.

## Arguments

| Name | Required | Valid Values | Description |
|---|---|---|---|
| *op* | yes | `cleanupResult` | |

## Return Values and Side Effects

A WavesetResult object containing the result of the cleanup.

# checkDeProvision Provision Workflow Service

Determines if an account needs deprovisioning before deletion.

## Arguments

| Name | Required | Valid Values | Description |
|---|---|---|---|
| *op* | yes | `checkDeProvision` | Identifies the name of the Identity Manager user to check for deprovisioning need. |
| user.waveset.accountId or accountId | yes | | Indicates the name of the user to check for deprovisioning need. |

## Return Values and Side Effects

A WavesetResult object containing the result of the cleanup.

# createResourceObject Provision Workflow Service

Creates a resource object (for example, a group).

## Arguments

| Name | Required | Valid Values | Description |
|---|---|---|---|
| *op* | yes | `createResourceObject` | |
| *object* | no | `GenericObject` | If not specified, the service looks for an argument named after the `ObjectType`. For example, if a resource supports managing a group `ObjectType`, and if the object argument is not supplied, then the service will expect the resource object to be in an argument called group. Preferred practice is to use the object argument. |
| *objectType* | yes | | Identity Manager resources can specify object types that they manage. This is specified by the `ObjectType` element list defined in the resource's prototype XML. This argument is the name of one of those `ObjectTypes`. |
| *resourceId* | yes | | Specifies the object ID for the resource on which to create the object. |

## Return Values and Side Effects

A WavesetResult object containing the result of the object creation.

# deleteResourceAccount Provision Workflow Service

Deletes a resource account.

## Arguments

| Name | Required | Valid Values | Description |
|---|---|---|---|
| *op* | yes | `deleteResourceAccount` | |
| *user.waveset.accountId* or *accountId* | yes | | Indicates the name of the Identity Manager user to delete. |
| *resource* | yes | | Identifies the name of the resource whose account needs to be deleted. |

## Return Values and Side Effects

A WavesetResult object containing the result of the account deletion.

# deleteResourceObject Provision Workflow Service

Deletes a resource object (for example, a group).

## Arguments

| Name | Required | Valid Values | Description |
|---|---|---|---|
| *op* | yes | `deleteResourceObject` | |
| *objectType* | yes | | Identity Manager resources can specify object types that they manage. This is specified by the `ObjectType` element list defined in the resource's prototype XML. This argument is the name of one of those `ObjectTypes`. |
| *object* | no | `GenericObject` | If not specified, the service looks for an argument named after the `ObjectType`. For example, if a resource supports managing a group `ObjectType`, and if the object argument is not supplied, then the service will expect the resource object to be in an argument called group. Preferred practice is to use the object argument. |
| *resourceId* | yes | | Specifies the object ID for the resource on which to delete the object. |

## Return Values and Side Effects

A WavesetResult object containing the result of the object creation.

# deProvision Provision Workflow Service

Deletes an Identity Manager account and/or resource accounts.

## Arguments

| Name | Required | Valid Values | Description |
|---|---|---|---|
| *op* | yes | deProvision | |
| *accountId* | yes | | Identifies the name of the Identity Manager user to deprovision. |
| *subject* | no | | Specifies the effective subject for the call. If not supplied, the task's subject is used. If the value of this argument is none, then no authorization is performed. |
| *options* | no | | If not supplied, specific arguments below are used. If supplied, any specific arguments below override the same argument contained in this options map. |

## Return Values and Side Effects

A WavesetResult object containing the result of the deprovision.

# deleteUser Provision Workflow Service

Deletes an Identity Manager user.

## Arguments

| Name | Required | Valid Values | Description |
|---|---|---|---|
| *op* | yes | `deleteUser` | |
| *subject* | no | | Identifies the effective subject for the call. If not supplied, the task's subject is used. If the value of this argument is none, then no authorization is performed. |
| *user.waveset.accountId* or *accountID* | yes | | Specifies the name of the Identity Manager user to approve. |
| *options* | no | Map | A value map of option name/option value pairs. If not supplied, the specific arguments given below are used. If supplied, any specific arguments below will override the same argument contained in this options map. |
| *user.waveset.accountId* or *accountID* | yes | | Specifies the name of the Identity Manager user to delete. |
| adminName | no | | Specifies the name of the administrator performing the deletion. |
| loginAppName | no | | Identifies the login application name. |
| force | no | true/false | If `true`, the user will be deleted regardless of whether there are resource accounts that need to be deprovisioned or not. |

## Return Values and Side Effects

A WavesetResult object containing the result of the deletion of the user.

# disable Provision Workflow Service

Disables an Identity Manager account and/or resource accounts.

## Arguments

| Name | Required | Valid Values | Description |
|---|---|---|---|
| *op* | yes | `disable` | |
| *subject* | no | | The effective subject for the call. If not supplied, the task's subject is used. If the value of this argument is `none`, then no authorization is performed. |
| *options* | no | | A value map of option name/option value pairs. If not supplied, specific arguments below are used. If supplied, any specific arguments below will override the same argument contained in this options map. |
| *accountId* | no | | Identifies the name of the Identity Manager user to disable. |
| *adminName* | no | | Specifies the name of the administrator performing the operation. |
| *loginAppName* | no | | Identifies the login application name. |
| *doLighthouse* | no | true/false | Indicates whether or not to disable the Identity Manager account. |
| *doResources* | no | true/false | Indicates whether or not to disable the user's resources. |
| *doAuthenticators* | no | true/false | If `true`, disables all pass-through authentication resources. |

## Return Values and Side Effects

A WavesetResult object containing the result of the disable.

# enable Provision Workflow Service

Enables an Identity Manager account and/or resource accounts.

## Arguments

| Name | Required | Valid Values | Description |
|---|---|---|---|
| *op* | yes | `enable` | |
| *subject* | no | | Indicates the effective subject for the call. If not supplied, the task's subject is used. If the value of this argument is `none`, then no authorization is performed. |
| *options* | no | Map | A value map of option name/option value pairs. If not supplied, specific arguments below are used. If supplied, any specific arguments below will override the same argument contained in this options map. |
| *accountId* | no | | Identifies the name of the Identity Manager user to enable. |
| *adminName* | no | | Indicates the name of the administrator performing the operation. |
| *loginAppName* | no | | Specifies the login application name. |
| *doLighthouse* | no | true/false | Indicates whether or not to enable the Identity Manager account. |
| *doResources* | no | true/false | Indicates whether or not to enable the user's resources. |
| *doAuthenticators* | no | true/false | If `true`, enables all pass-through authentication resources. |

## Return Values and Side Effects

A WavesetResult object containing the result of the enable.

# getApprovals Provision Workflow Service

Determines the lists of approvals for the assigned role, organization, and resources for an existing account.

## Arguments

| Name | Required | Valid Values | Description |
|---|---|---|---|
| *op* | yes | `getApprovers` | |
| *user* | yes | `GenericObject` | The `view representing the user`. |

## Return Values and Side Effects

If the `getApprovals` workflow is successful, a `GenericObject` describing the approvals necessary for the user is returned in the *`approvals`* variable. If any error occurred, `WF_ACTION_ERROR` will be `true`, and the WavesetResult will contain the error.

# lockOrUnlock Provision Workflow Service

Locks or unlocks a specified user if the Lighthouse Account Policy associated with the user specifies a lock expiration time.

The user is unlocked when one of these conditions is met:

- a user who is assigned the Unlock User capability unlocks the user's Lighthouse account
- the current date and time is later than the user's lock expiration date and time, if one was set

While a user is locked, the user cannot log in by any login application interface (for example, the User or Administrator interfaces or the Business Process Editor). By default, the `lock` service is called by the Failure activity of the Password Login process that is invoked when the number of failed password login attempts exceeds those allowed by the user's Lighthouse Account policy.

## Argument

`lock` -- (Boolean) Indicates whether the user is locked (`true` indicates locked).

## Return Values and Side Effects

This method returns null if it is a lock request. If it is an unlock request, this method returns a WavesetResult that contains the unlock results.

# notify Provision Workflow Service

Sends a notification, which is almost always an email.

## Arguments

| Name | Required | Valid Values | Description |
|---|---|---|---|
| *op* | yes | notify | |
| *type* | yes | | Indicates the type of notification. If not entered, email type will be used. |
| *various other arguments* | no, but should be supplied. | | Identifies the arguments passed to the notifier. Map of name/value pairs. |

## Return Values and Side Effects

A WavesetResult object containing the result of the notify.

# provision Provision Workflow Service

Creates a new Identity Manager account and (optionally) resource accounts.

## Arguments

| Name | Required | Valid Values | Description |
|---|---|---|---|
| *op* | yes | `Provision` | |
| *user* | yes | `GenericObject` | Specifies the `GenericObject` that contains the view that is to be provisioned |
| various provisioning options | no | | Identifies the options to specify provisioning characteristics. See the javadocs for `com.waveset.provision.ProvisioningOptions`. |

## Return Values and Side Effects

A WavesetResult object containing the result of the provision.

# questionLock Provision Workflow Service

Locks the user but does not set a lock expiration time or date.

A user who is locked with this service can be explicitly unlocked by an administrator who has the Unlock User capability. If an administrator does not unlock this user, his account is automatically unlocked the next time the user's password is changed or reset. While a user is locked by this service, he can still log in by any other Identity Manager application interface except the Forgot My Password page.

By default, the `questionLock` service is called by the Failure activity of the Question Login process, which is invoked when the number of failed question login attempts exceeds those allowed by the user's Lighthouse Account Policy.

## Return Values and Side Effects

Returns null.

# reject Provision Workflow Service

Records the rejection of a resource account.

## Arguments

| Name | Required | Valid Values | Description |
|---|---|---|---|
| *op* | yes | `reject` | |
| *user.waveset.accountId* or *accountId* | yes | | Identifies the name of the Identity Manager user to reject. |
| *options* | no | `Map` | Indicates the options to specify provisioning characteristics. See the javadocs for `com.waveset.provision.ProvisioningOptions` |

## Return Values and Side Effects

A WavesetResult object containing the result of the rejection.

# reProvision Provision Workflow Service

Updates an existing Identity Manager account.

## Arguments

| Name | Required | Valid Values | Description |
|---|---|---|---|
| *op* | yes | `reProvision` | |
| *user* | yes | `GenericObject` | Identifies the `GenericObject` that contains the view that is to be re-provisioned. |
| *various provisioning options* | no | | Indicates the options to specify provisioning characteristics. See the javadocs for `com.waveset.provision.ProvisioningOptions`. |

## Return Values and Side Effects

A WavesetResult object containing the result of the reprovision.

# runResourceAction Provision Workflow Service

Executes a resource action on the specified resource adapter for a resource.

You can specify the resource action either directly (using `resourceAction`) or by ID (using `resourceId`). If `resourceAction` correctly references a resource action, it overrides the ID specified in `resourceId`.

This method does not contain an authorization check. All the rules governing resource actions apply.

## Arguments

| Name | Required | Valid Values | Description |
|---|---|---|---|
| `resourceId` | | | Identifies the resource to use. |
| `resourceAction` | | | (Optional) Specifies the resource action to run. This value overrides the ID specified in `resourceId`. |
| `resourceActionId` | | | Identifies by ID the resource action that the adapter runs. |
| `resourceActionArgs` | | | Provides a map of the arguments that this method should use. These arguments are passed to the script in the action as environment values, where the names of the variables are `RESACTIONARG_<key>` and the value is the value that corresponds to that key. Keys and values should be strings. |

**Note** The variables are accessed as described above with one exception: HostAccessResourceAdapter subclasses that support ACTIONS use JavaScript for running actions, and the args are passed in a global variable called `additionalArgs`.

## Return Values and Side Effects

A WavesetResult object containing the results of the executed action

# updateResourceObject Provision Workflow Service

Updates a resource object (for example, a group).

## Arguments

| Name | Required | Valid Values | Description |
|---|---|---|---|
| *op* | yes | `updateResourceObject` | |
| *objectType* | yes | | Identity Manager resources can specify object types that they manage. This is specified by the `ObjectType` element list defined in the resource's prototype XML. This argument is the name of one of those `ObjectTypes`. |
| group | yes | `GenericObject` | Specifies the `GenericObject` to update on the resource. If not specified, the service looks for an argument named after the `ObjectType`. For example, if a resource supports managing a `group` ObjectType. If the *`object`* argument is not supplied, then the service expects the resource object to be in an argument called *`group`*. Preferred practice is to use the *`object`* argument. |
| resourceId | yes | | Specifies the object ID for the resource on which to update the object. |

## Return Values and Side Effects

A WavesetResult object containing the result of the object update.

# Type Names

These are the valid type names that are used by Identity Manager:

- AdminGroup
- Administrator
- Application
- AttributeDefinition
- AuditConfig
- AuditLog
- AuditLogPrunerTask
- AuditQuery
- Configuration
- DataStore
- DataType
- Discovery
- Encryptionkey
- EmailTemplate
- Event
- Extract
- LoadConfig
- Log
- LoginConfig
- NotifyConfig
- ObjectGroup
- Policy
- RemedyConfig
- ReportCounter
- ReportLog
- Resource
- ResourceAccount
- ResourceObject
- Role
- RoleAttribute

- Rule
- TaskDefinition
- TaskInstance
- TaskResult
- TaskResultPage
- TaskSchedule
- TaskTemplate
- TestItem
- UsageReport
- User
- UserExternalAttribute
- WorkItem

# Right Names

These are the valid right names that are used by Identity Manager:

- Approve
- Bypass Verify
- Change Password
- Configure
- Connect
- Create
- Delete
- Deprovision
- Disable
- Disconnect
- Enable
- Execute
- Export
- Import
- List
- Modify
- Provision
- Reject
- Rename
- Reset Password
- Unlock
- View

# Action Names

These are the valid action names that are used by Identity Manager:

- Approve
- CancelReconcile
- ChangePassword
- Configure
- Connect
- Create
- CredentialsExpired
- Delete
- Deprovision
- Disable
- Disconnect
- Enable
- Load
- Login
- Logout
- Modify
- NativeChange
- Provision
- Reject
- RequestReconcile
- Terminate
- View

# 3 Identity Manager Forms

This chapter describes how you can customize the appearance and behavior of selected pages in Sun Java™ System Identity Manager Administrator and User Interfaces by customizing the *forms* that define these pages.

## Topics in this Chapter

This chapter is organized into the following sections:

- **Understanding Forms** – Introduces basic form concepts and describes how forms are integrated into the Identity Manager system.
- **Customizing Forms** – Describes form programming syntax and logical guidelines to use when working with forms and provides examples of different form elements. Also describes how to use the Business Process Editor (BPE), a standalone graphical tool, to edit forms.
- **Testing Your Customized Form** – Provides techniques to use when verifying your form syntax and tracing field logic in your custom forms.

## Related Chapters

See these related chapters:

- *Views* – Identity Manager forms interact with an internal Identity Manager data structure called the user view. When customizing a form, you can call view attributes.
- *HTML Display Components* – You use the HTML component language to create field definitions when editing a form.
- *XPRESS Language* – You use expressions to include logic in your forms.
- *Introduction to the Business Process Editor* in *Identity Manager Deployment Tools*– Introduces the Business Process Editor (BPE) and describes how to start the tool, set editor options, save, and debug forms.

# Understanding Forms

To customize Identity Manager's Web-based user interface appearance and function, you must modify the form associated with the web page you want to edit.

The term *form* can describe both the web page where users enter information and the object that contains rules about how to display data in the view. Throughout this guide, the term *form* typically refers to the object that contains rules about how to display data in the view.

This section covers the following topics:

- What are Forms?
- Why Edit Forms?
- Identity Manager Pages that Use Forms
- Commonly Edited Forms
- How Do Forms Work?

## What Are Forms?

A *form* is an object associated with a page that contains rules about how the browser should display user view attributes on that page. Forms can incorporate business logic and are often used to manipulate view data before it is presented to the user.

For example, to create a new user account, you use the Create User page, in which you enter information about the new user. This page is generated using an object (a form) in the Identity Manager repository named Tabbed User Form. This form specifies which fields are visible on the Create User page and which HTML form element (for example, text box, check box, or select box) is used to represent each field. This form also specifies additional logic for disabling fields, populating empty fields with default values, and calculating field values from the values of other fields.

### What Forms Control

Forms control:

- **Layout and display characteristics of the page**

  Forms are composed of fields. Visible field types include simple text boxes, radio buttons, and selection boxes with multiple values. Fields can also have values based on other fields and can be either read-only or be hidden from view.

- **Data that is used on the page**

  Data can be captured dynamically from a resource or be calculated from other fields. With the Identity Manager expression language called XPRESS, field data can be calculated, concatenated, and logically evaluated.

- **Data that is coming into the system**

  Forms can be the interface from web pages as well as from noninteractive systems such as ActiveSync resources. In this role, the form has no visual fields, but still provides rules to set default values and other field values.

  For example, the Full Name field might not be visible to the administrator using the page, but can be set based on the values that the user enters into the First Name, Middle Name, and Last Name fields. Populating fields from other fields reduces the data entry that users and administrators must perform, consequently reducing potential data entry errors. Likewise, by providing option menus in the place of text input fields, an administrator can select a department from a list instead of entering the department name. For information on the specific HTML components that define the default Identity Manager forms, see *HTML Display Components*.

- **Identity Manager background processing**

  Forms are also used within Identity Manager in the background processing. For example, forms can work in conjunction with resource adapters to process information from an external resource before storing it in the Identity Manager repository.

  When creating forms to manipulate data in the background, you focus primarily on encoding logic because the appearance is irrelevant in forms that are not visible to users. For more information on using hidden (nonvisible) components, see the section titled *Using Hidden Components*.

## Sample Form and Displayed Browser Page

The following Identity Manager page presents four fields to a user:

- account ID
- First Name
- Last Name
- Full Name

The user enters information into the First Name and Last Name fields, and the system calculates the value of the last field, Full Name, from the values of these two fields.

## Sample Browser Page

The rendering of this form in the browser is shown in the following illustration.

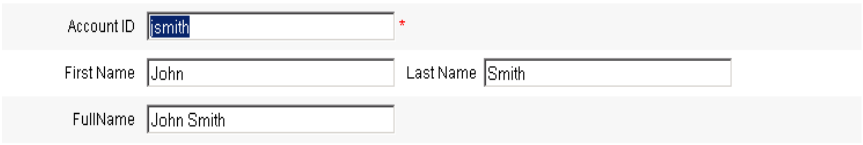



Figure 1. Form for Entering User First Name and Last Name

## Sample Form

The following XML example defines the form fields that are used to render the browser fields in the preceding illustration. It specifies how the user's full name is built out of the information entered into the First Name and Last Name fields.

```
<Field name='waveset.accountId'/>
   <Display class='text'>
     <Property name='title' value='AccountID'/>
   </Display>
</Field>
<Field name='global.firstname'>
   <Display class='Text'>
      <Property name='title' value='First Name'/>
      <Property name='size' value='32'/>
      <Property name='maxLength' value='128'/>
   </Display>
</Field>
<Field name='global.lastname'>
   <Display class='Text'>
      <Property name='title' value='Last Name'/>
      <Property name='noNewRow' value='true'/>
      <Property name='size' value='32'/>
      <Property name='maxLength' value='128'/>
   </Display>
</Field>
<Field name='global.fullname'>
   <Display class='Text'>
      <Property name='title' value='FullName'/>
      <Property name='size' value='32'/>
      <Property name='maxLength' value='32'/>
   </Display>
      <Expansion>
         <concat>
            <ref>global.firstname</ref>
            <s> </s>
            <ref>global.lastname</ref>
```

```
        </concat>
      </Expansion>
</Field>
```

## Why Edit Forms?

Why customize the default Identity Manager pages, which already provide all the fields that you need to perform actions within the product? Customizing the default forms allows you to better enforce your company's policies and processes:

- **Preserve privacy by limiting the amount of user account information displayed on the screen**. You may not want to present all of the information available for a user account depending on who is viewing the information because of concerns for privacy or to reduce the distraction from nonessential information.
- **Provide context-specific help on individual fields**. This can reduce confusion and calls into your help desk.
- **Reduce the distraction of nonessential information for users performing a specific task**. Typically, the most effective way to present information is to display only the fields you need to accomplish the current task.

Customizing the default fields in Identity Manager forms allows you to extend and customize the application for your environment. Specifically, you can customize the default forms to:

- **Address the specific needs of the users in your organization**. This is particularly important when several different types of administrators must access different portions of the same view data and should not view all data attributes. For example, a human resources administrator may need to access a different subset of user account attributes than a help desk administrator.
- **Control the display and content of the user account attributes**, particularly attributes displayed on the Create User and Edit User pages. These pages contain most of the attributes that need to be controlled.
- **Define default values for user view attributes** and their associated attributes. For example, you could define a default home directory for a user instead of the administrator having to key in the path.
- **Pre-process user view attributes before they are displayed.** For example, department or division codes that are stored as acronyms or by numeric ID in your resource can be represented with more human-readable full names to your user.
- **Post-process user view attributes data entry**. For example, you can automatically create a mail account based on the value of a location field.

- **Control screen real estate by positioning multiple fields on one line**. By customizing the arrangement of fields in an Identity Manager form, you can make it more closely resemble a printed form or pre-existing web form.
- **Define rules** for the way hidden attributes are calculated. For example, a user's email address can be calculated to be the user's first name, a period, their last name, then the mail domain: `joe.user@sun.com`

## Example Scenario

Forms are especially useful in environments where people with varying needs and purposes must access the same data.

For example, you can create a form that hiring managers at your company will use to create a new employee account. The default Tabbed User Form displays more information than the hiring managers need. Rather than displaying all 99 fields in a distractingly busy form that might complicate the user's task, you can create a form in which the hiring managers must fill in only 10 attribute fields and the other 89 attributes are set based on rules that you, the administrator, define.

## Identity Manager Pages that Use Forms

Identity Manager forms are typically classified into one of two categories:

- **Forms that drive the GUI**. These forms, which can be part of either the Identity Manager Administrator or Identity Manager User Interface, include the pages that users use when:
  - Changing passwords
  - Performing self-service
  - Administrative tasks that involve account creation, system configuration, and workflow tasks.

  You can use the default forms that ship with Identity Manager as springboards for creating your own custom forms. While you will probably want to copy and directly edit only a subset of these forms (see the section titled *Commonly Edited Forms*), you can peruse other forms for examples of how to encode particular attributes or behaviors.
- **Forms that perform background-processing on information being imported into Identity Manager** from an external resource. This chapter does not discuss how to use forms for background processing. For example, as part of the process of reading information from a PeopleSoft database into Identity Manager, a form checks employee status on incoming records. If the employee status is not active (A), the form defines a field that disables the Identity Manager account for that user.

The following table shows some of the Identity Manager pages that use forms of the first type. Use this table to identify the form that controls the display characteristics of the page you want to edit.

| Page You Want to Edit | Associated JSP | Associated Form |
|---|---|---|
| Create/Edit User | `account/modify.jsp` | Tabbed User Form |
| Change User Account Attributes | `user/changeAll.jsp` | End User Form |
| Welcome | `user/anonmmain.jsp` | Anonymous User Menu |
| Edit Work Item | `approval/itemEdit.jsp` | Approval Form |

## Commonly Edited Forms

Of the default forms that ship with Identity Manager, you will probably edit one of the following five forms:

- End User Menu Form
- Anonymous User Menu Form
- Tabbed User Form
- End User Form
- Approval Form

These commonly edited forms control the creation and modification of users and the display of the main menu that the user sees. They are described in greater detail in the following sections.

### End User Menu Form

End User Menu Form controls the display of the main menu in the Identity Manager User interface. Typically, this form contains links for changing the user's password, editing account attributes, and changing answers to authentication questions.

You can customize End User Menu Form to add links to launch special workflow processes that are accessible to the user (for example, a process to request access to a system).

For example, to present the End-User Test Process as a link to click from the end user pages, add the following:

```
<Configuration id='#ID#Configuration:EndUserTasks' name='End User
Tasks'>
   <Extension>
     <List>
        <List>
        <String>End-User Test Process</String>
        <String>An example end-user workflow</String>
        </List>
     </List>
```

The Identity Manager User Interface displays a list of self-service processes for selection. This is expected to be a list of lists. The first element of the sublist displays the process name, and the second element describes what the process does.

## Anonymous User Menu Form

Anonymous User Menu Form controls the display of the main menu in the Identity Manager User interface when an unknown user logs in.

Identity Manager uses the anonymous end user pages for users who are not defined in the system through the process of *user self-provisioning*. For example, a Identity Manager administrator can set up pass-through authentication for an Active Directory resource. As a result, any person who has an Active Directory account can log in to the Identity Manager User interface. You can customize those pages so that when a user who does not have a Identity Manager account logs in, an Identity Manager user object is created and the Active Directory resource is added. Subsequently, through a series of questions, the system can set up the user's role, organization, and other resources.

You can customize Anonymous User Menu Form to launch workflow processes to request services before an Identity Manager user exists.

## Tabbed User Form

Tabbed User Form is the default form used for user creation and modification in the Identity Manager Administrator Interface. You can customize a copy of this form by extending it with a form of your design.

**Tip** Do not directly edit the Tabbed User Form. Instead, Sun recommends that you make a copy of this form, give it a unique name, and edit the renamed copy. This will prevent your customized copy from being overwritten during service pack updates and upgrades.

Customize your copy of Tabbed User Form to:

- Restrict the number of attributes that are displayed on the Edit User page. By default, this page displays every attribute that is defined on the schema map for a resource, which can result in an overwhelming list of attributes for a hiring manager to fill out.
- Set the default field types to more helpful select boxes, checkboxes, and multi value fields. By default, every attribute defined on a resource assigned to a user will appear on the Create User and Edit User pages as a text box (or as a checkbox for Boolean values).
- Include additional forms to allow common forms to be used on multiple pages.

### Basic Fields

Tabbed User Form contains these fields:

- `accountId`
- `firstname`
- `lastname`
- `role`
- `organization`
- `password`
- `confirm password`
- `email`
- `resource list`
- `application list`
- `MissingFields`

**Note** The MissingFields field is not actually a field but an element that indicates to the form generator that it should automatically generate text fields in the global namespace for all attributes that can be pushed to the assigned resources that are not explicitly declared in the Tabbed User Form. Remove this field if you do not want the form locator to explicitly generate these fields.

By default, every attribute defined on a resource that is assigned to a user appears on the Create User and Edit User pages as a text box (or checkbox for Boolean values).

## End User Form

End User Form controls the page that the system displays when a user selects **Change Other Attributes** from the `/user/main.jsp` on the Identity Manager User interface. From this page, a user can change his password, authentication questions, and email address.

You can customize End User Form to grant users control over other fields, such as those that handle phone numbers, addresses, and physical office locations. For example, you can add a field through which users can request access to additional NT groups through the Identity Manager pages.

### Approval Form

Approval Form controls the information that is presented to a resource, role, or organization owner when he is designated an approver of user requests. By default, this page displays a set of read-only fields that contain the name of the administrator that started the process. It also displays information about the user, including the account ID, role, organization, and email address.

This form ensures that the resource owner gets a last chance to change a user value before the user is created. By default, approving a user displays all the user attributes in read-only fields.

You can customize Approval Form to:

- Add and remove information about a user.
- Assign the approver the ability to edit this information so that he can modify the information entered on the initial user form.
- Create your own approval forms for different purposes. For example, you can create different approval forms for use when an administrator or resource owner initiates account creation or deletes a user.

## How Do Forms Work?

Various factors affect how the browser displays a form. However, form behavior within the browser is primarily determined by:

- **View associated with the form**. All forms are used with views. The most common view used with forms is the user view. The view defines the data that is available when the form is evaluated.
- **Undefined attributes**. The Tabbed User Form provides a mechanism for automatically generating text fields to edit resource account attributes for which fields are not explicitly defined. You can disable this feature in the form.
- **How forms interact with other Identity Manager components**. This includes the process by which Identity Manager evaluates the form, or *form evaluation*. All form-driven pages are processed similarly. For an overview of how Identity Manager evaluates a form, see *Form Evaluation*.
- **Display components used in the form**. Form fields can be associated with a display component that determines how the field is displayed in the browser.

## User View and Forms

The *user view* is a data structure that contains all available information about an Identity Manager user. It includes:

- Attributes stored in the Identity Manager repository
- Attributes fetched from resource accounts
- Information derived from other sources such as resources, roles, and organizations

Views contain many attributes, and a *view attribute* is a named value within the view (for example, `waveset.accountId` is the attribute in the user view whose value is the Identity Manager account name).

Most form field names are associated with a view attribute. You associate a field with a view attribute by specifying the name of the view attribute as the name of the form field. For more information, see *Defining Field Names*.

For more information on the user view, including a reference for all attributes in the user view, see the chapter titled *Views*.

## Undefined Attributes

When a resource or role is assigned to a user through the administrative interface, a refresh occurs. The new resource account attributes are then defined in the User view. `<FormRef name = 'Missing Fields'/>` in the Tabbed User Form indicates to the form generator that text fields should be generated for any resource account attributes that do not have a corresponding field explicitly defined in the form. To disable this feature in the Tabbed User Form, delete `<FormRef name = 'Missing Fields'/>`.

## Form Evaluation

How the system processes a form helps determine the behavior of the form in the browser. All form-driven pages are processed similarly, as described below:

1. A **page is requested** from the Identity Manager User or Administrator Interface.
2. The **interface requests a view from the server**. A *view* is a collection of named values that can be edited. Each view is associated with a form that defines how the values in the view are displayed to the user.
3. The **server assembles a view** by reading data from one or more objects in the repository. In the case of the user view, account attributes are also retrieved from resources through the resource adapter.

4. **Derivation expressions are evaluated**. These expressions are commonly used to convert cryptic, encoded values from the resource into values that are more meaningful to the user. Derivations are evaluated when the form is first loaded or data is fetched from one or more resources.
5. **Default expressions are evaluated**. These fields are set to the default value if the field is null.
6. **HTML code is generated**. The system processes view data and the form to produce an HTML page. During this processing, the `allowedValues` properties within expressions are evaluated to build `Select` or `MultiSelect` HTML components.
7. The page is presented in the browser, and the user can edit the displayed values. During editing, the user typically modifies fields, which can result in a refresh or recalculation of the page. This causes **the page to be regenerated**, but the system does not yet store the edited data in the repository.
8. **Modified values are assimilated back into the view**. When a refresh event occurs, the interface receives values for all the form fields that were edited in the browser.
9. **Expansion expressions are evaluated**. This can result in additional values being placed into the view. Expansion rules are run whenever the page is recalculated or the form is saved.
10. **The view is refreshed**. The interface asks the server to refresh the view and provides the current set of edited values. The server may insert more values into the view by reading data from the repository or the resources.
11. **Derivation expressions are evaluated**. Typically, derivation expressions are not evaluated when a view is refreshed. In some complex cases, the system can request derivations after the refresh.
12. The system processes the refreshed view and form and **builds another HTML page, which is returned to the browser**. The user sees the effects of the refresh and continues editing. The user can cause the view to be refreshed any number of times (repeating steps 7 through 12 each time) until the user either saves or cancels the changes.
13. A. If the edit is canceled, all the data accumulated in the view is discarded, and the server is informed. As a result, the server can release any repository locks, and control passes to a different page.

    B. If the edit is saved, the interface receives the values that have been modified and assimilates them into the view (see step 8).
14. **Validation expressions** are evaluated. If field values do not meet required specifications, then an error is presented and the field values can be corrected. Once the changes have been made, the process returns to step 13.
15. **Expansion expressions** are evaluated one last time (see step 9).

16. If the server saves the view, this typically results in the **modification of one or more objects in the repository**. With user views, resource accounts may also be updated.

Several of the preceding steps require iteration over all the fields in the form. These include the evaluation of `Derivation` expressions, the evaluation of `Default` and `Validation` expressions, the generation of HTML, and the evaluation of `Expansion` expressions. During all field iterations, `Disable` expressions are evaluated to determine if this field should be processed. If a `Disable` expression evaluates to true, the field (and any nested fields it contains) is ignored. See *Defining Field Elements* in this chapter for more information on these special types of expressions.

# Customizing Forms

After familiarizing yourself with the default operation of the Identity Manager product, you can identify pages you'd like to customize.

1. Consult the section titled *Commonly Edited Forms* for a list of editable pages and their corresponding forms.
2. To edit a form, launch the Business Process Editor (BPE) and select Open Repository Object. Select the form you want to edit from the popup dialog that is displayed.

This section covers the following topics:

- **Overview of Customization.**
- **Using the BPE to Create and Edit Forms**. Gives an overview of how to use the BPE to view and edit forms.
- **Additional Customization-Related Topics**
  - Form Structure
  - What is a Form Field?
  - Guidelines for Structuring a Form Defining Field Display
  - Optimizing Expressions in Form Fields
  - Adding Guidance Help to a Form

## Overview of Customization

You can customize a form to make it more user-friendly, change its display characteristics, or include logic for processing field data.

### Basic Steps

The basic steps for customizing any form in the Identity Manager system include:

- **Selecting a form to customize**. Describes how to identify which form to customize.
- **Editing and saving the form**. Presents basic information about using the BPE to modify the default end user and administrator forms shipped with the product.
- **Testing your changes**. Suggests guidelines for testing your changes before loading them into your production environment and turning on error logging.

### Typical Tasks

When you edit a form, you typically perform the following tasks:

- **Add and remove fields in the form.** Typical tasks include removing some default fields or adding additional fields that have been customized for your environment.
- **Define how a field is displayed within a form**. This requires using a library of HTML components shipped with Identity Manager. For information on editing a field's display characteristics, see the section titled *Defining Fields Display*.
- **Set the logical expressions that define the field's value**. To do this, you must create logical expressions using the XPRESS language. For information on working with XPRESS, see *XPRESS Language.*

## Using the Business Process Editor (BPE)

The Business Process Editor (BPE) application is a standalone, Swing-based Java application that provides a graphical and forms-based view of Identity Manager forms, rules, workflows, and email templates. It also permits the display of view attributes for reference while you customize forms and rules.

For guidelines on starting the BPE and using keyboard shortcuts, see *Introduction to the Business Process Editor*. This chapter also discusses how to use the BPE debugger to debug forms, workflows, and rules.

To display information about forms in the BPE:

- Start the application
- Specify the user or object for which you want to retrieve information and the type of attributes

## Loading a Form

To load a form into the Business Process Editor (BPE):

1. Select **File → Open Repository Object** from the menu bar.
2. If prompted, enter the Identity Manager Configurator name and password in the displayed login dialog, and then click **Login**.

The BPE displays a list of items to select. The specific objects displayed depend upon the contents of your repository and your access permissions.

The BPE displays a list of items to select, which include:

- Workflow tasks 
- Workflow subprocesses 
- Email templates 
- Forms 
- Views 
- Rules 
- Libraries 
- Configuration Objects 
- Generic Objects 
- MetaView 

**Note** Items displayed may vary for your Identity Manager implementation.

3. Select a form, and then click **OK**.

# Getting Around in the Business Process Editor

Before you start customizing a form, you should know how to work, view and enter information, and make selections in the BPE.

The BPE interface includes a menu bar and dialogs for selections. When editing a form, the primary display is divided into three panes: the *tree view*, the *property* view, and the *graphical view*.

## Tree View

The tree view (in the left interface pane) shows a hierarchical view of a task, form, view, or rule. It lists each element in order, nesting sub-elements under their parent.

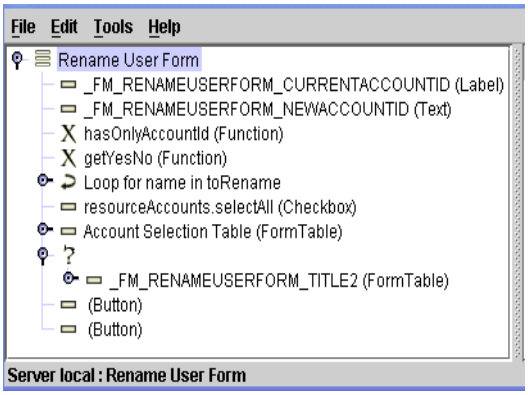



Figure 2. Tree View Display of Form (Expanded)

The tree view displays form elements as:

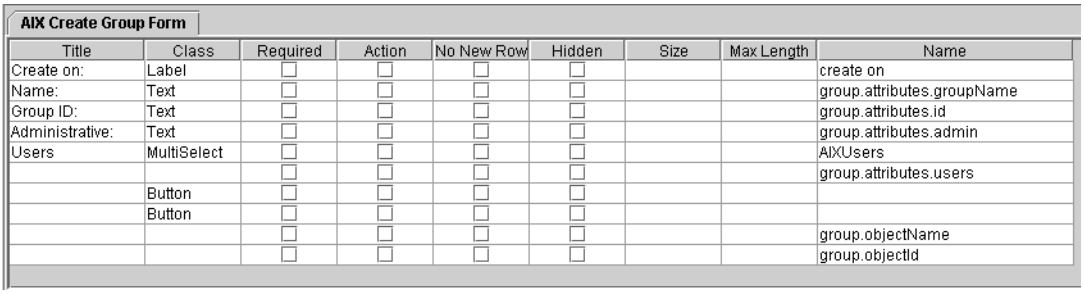
- Field
- Loop
- Form reference
- Field reference
- Expression

Using the BPE to create these form elements is covered in later sections.

## Property View

The property view appears in the upper right interface pane, and provides a schematic representation of form elements. Each entry in the property view represents a form element.

You can change form elements in the Property view, then select Refresh from the action menu. The Graphical pane of the main display changes to reflect your changes if the changes affect the visible features of the form.



AIX Create Group Form

| Title | Class | Required | Action | No New Row | Hidden | Size | Max Length | Name |
|---|---|---|---|---|---|---|---|---|
| Create on: | Label | ☐ | ☐ | ☐ | ☐ | | | create on |
| Name: | Text | ☐ | ☐ | ☐ | ☐ | | | group.attributes.groupName |
| Group ID: | Text | ☐ | ☐ | ☐ | ☐ | | | group.attributes.id |
| Administrative: | Text | ☐ | ☐ | ☐ | ☐ | | | group.attributes.admin |
| Users | MultiSelect | ☐ | ☐ | ☐ | ☐ | | | AIXUsers |
| | | ☐ | ☐ | ☐ | ☐ | | | group.attributes.users |
| | Button | ☐ | ☐ | ☐ | ☐ | | | |
| | Button | ☐ | ☐ | ☐ | ☐ | | | |
| | | ☐ | ☐ | ☐ | ☐ | | | group.objectName |
| | | ☐ | ☐ | ☐ | ☐ | | | group.objectId |

Figure 3. Property View of Form

This view provides the following information about form elements:

| Property | Description |
|---|---|
| Title | Specifies the text that displays adjacent to the form field. |
| Class | Identifies the HTML display class to which the element belongs. |
| Required | Identifies whether the element is required to process the form. This field must have a non-null value upon submission. When set, results in a red asterisk appearing to the right of the field. Message text at the bottom of the form indicates that red asterisk denotes fields that must have a value for submission to proceed. |
| Action | When set, a change causes the page to refresh any `Select` or `MultiSelect` controls. In the Identity Manager Administrator Interface, this causes the underlying view to be refreshed. Role selection exemplifies this behavior. When a new role is selected in the Tabbed User Form, the view is refreshed to reflect the resources that are assigned through that role during that edit session. After the view has been refreshed, resource account attributes on those newly assigned resources can be explicitly set. |
| No New Row | Used strictly for form layout. When true, forces the field to appear to the right of the proceeding field. For example, Name fields are examples where this is useful, where it is desirable to allow the user to enter the last name, first name, and middle initial from right to left, rather than down the page. |
| Hidden | Indicates the field should not be visible to the user. The field is typically used to set attribute values that are calculated form other fields, such as constructing the full name from a concatenation of first and last name. |
| size | Controls the character width of the control (text boxes). |
| maxLength | Specifies the character width of the control buffer (text boxes). Characters scroll if the user types in a string greater than the value specified by the `size` property. |
| Name | Identifies the name for this form field, typically a path expression in to the view that is used with this form. |

## Graphical View

The graphical view of the form appears in the lower right pane of the BPE display. It displays the form HTML as displayed by Identity Manager.

Right-clicking in this area of the display accesses the action menu that contains the following options:

- **Create a Shortcut** – Creates a shortcut to this HTML form file on your desktop.
- **Add to Favorites** – Adds this HTML form file to your list of Internet Explorer favorites.
- **View Source** – Displays form source. Right-click to display the action menu, then select View Source. A Notepad window displaying the source for the form as displayed in the graphical view.
- **View Encoding** – Displays options for various available language sets, including ISO and MS.
- **Print** – Opens a standard print dialog.
- **Refresh** – Refreshes the graphical display, updating it to display any element changes you've made in the current BPE session.
- **Export to Excel** – Opens an Excel spreadsheet.
- **Properties** – Displays information about general form characteristics such as size and creation and modification dates and, where relevant, security certificates.

## Form Dialogs

Each form and form element has an associated dialog that allows you to define the element type and its characteristics.

The major form dialogs include:

- Form dialog
- Form Element dialog
- Form ToolBox dialog

The following sections give general tips on working with form-related dialogs.

## Form Dialog

Use the Form dialog to view and set the essential characteristics of a form. Double-click the form name in the tree view to display this dialog.

Four views of information on the selected form are accessible from this dialog:

- Main tab (default)
- Includes tab
- Properties tab
- XML tab

## Main Tab

Click to display the Main view (typically displayed by default).

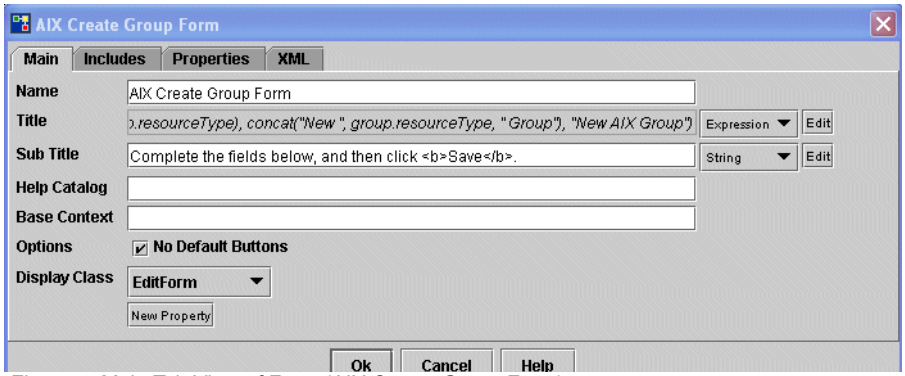



Figure 4. Main Tab View of Form (AIX Create Group Form)

Set the characteristics in the following table from the Main tab view.

| Field | Description |
|---|---|
| Name | Enter the name for this field. A *field name* is typically a path expression into the view that is being used with this form. All fields that display as editing components (such as text boxes, checkboxes, and selects) must have a name that specifies a view path. Fields that do not display as editing components (such as `SectionHead` and `Javascript`) do not require names. However, you can give non-editing fields names if they need to be referenced by another form through a Field reference. |
| Title | Enter a title for the field. This title displays adjacent to the field on the form. Select the data type of this element from the drop-down menu immediately adjacent to this field. To edit the text displayed in this field, click the adjacent **Edit** button. |
| Sub Title | (Optional) Specify text that Identity Manager can display beneath the form title. Select the data type of this element from the drop-down menu immediately adjacent to this field. To edit the text displayed in this field, click the adjacent **Edit** button. |
| Help Catalog | Enter the help key that associates guidance help with the field. This value is the name of an entry in an associated help catalog specified by the form. Specifying a help key causes an icon to appear to the left of the field. Moving the mouse over the icon causes the text referenced in the help catalog to display. |
| Base Context | (Not typically used in standard user forms.) Enter the base context to avoid the need to specify the full path in every field. *Base context* identifies the underlying Map (specifically, `com.waveset.object.Genericobject` and is typically named `user` or `userview`. In the administrative interface, the editing context is user, so the base context reference is left blank. In forms launched from manual actions, such as approvals, the workflow context is the context of the form. |

| Field | Description |
|---|---|
| Options | Select one or more display options for the field:<br>• **Required** -- Identifies whether the element is required to process the form. This field must have a non-null value upon submission. When set, results in a red asterisk appearing to the right of the field. Message text at the bottom of the form indicates that red asterisk denotes fields that must have a value for submission to proceed.<br>• **Button** -- Causes the field to display in a single, horizontal row at the bottom of the form. Otherwise, it displays on the next line of the form. This is most commonly set with fields that use the display class Button.<br>• **Action** -- When set, a change causes the page to refresh any `Select` or `MultiSelect` controls. In the Identity Manager Administrator Interface, this causes the underlying view to be refreshed. Role selection exemplifies this behavior. When a new role is selected in the Tabbed User Form, the view is refreshed to reflect the resources that are assigned through that role during that edit session. After the view has been refreshed, resource account attributes on those newly assigned resources can be explicitly set.<br>• **Library** -- Indicates that a field should only display when it is referenced, rather than when it is declared. This is useful when the order in which fields are evaluated on a form may differ from the order in which they are displayed to the user. |
| Default | Specify an expression to calculate a default value for the field. The default expression is called before the form is displayed if the current value for this field is null. |
| Derivation | Specify an expression to calculate the value of a field before it is displayed. It is similar to a Default expression, except that it is evaluated even if the current field value is non-null. The derivation expression is evaluated before the form is first displayed, and then again each time the form is refreshed. |
| Validation | Specify logic to determine whether a value entered in a form is valid. Validation expressions return null to indicate success, or a string containing a readable error message to indicate failure |
| Expansion | Specify an expression to calculate the value of the field after the form has been submitted. Expansion expressions are typically used with fields that are also marked hidden. Since hidden fields are not directly editable by the user, the value can be calculated with an Expansion expression. See *Hiding Fields* later in this chapter. |

| Field | Description |
|---|---|
| Disable | Specify an expression that, if evaluated to true, disables the field and any of its nested fields. A disabled field does not display on the form. It is commonly used to determine if a user has a specific type of resource. If the user does, the form then displays the appropriate fields for that resource. |
| Display Class | Identify the HTML component class used to render this form component in the browser. By default, the Display Class selection is EditForm. If the form is a link form (such as the End User menu), then select LinkForm from the Display Class options.<br>See the HTML Display Class table in *HTML Display Components*. |
| size | Controls the character width of the control (text boxes). |
| maxLength | Specify the maximum number of characters for this element. |

To edit a display property, double-click the item in the Properties area.

To delete a display property, click to select it, and then click **Delete**.

The properties you can edit are listed in the Main menu (`title`, `subTitle`, and `width`). The value of the property can be a simple string or computed by an expression. If you use an expression, click **Validate** to test it. Click **OK** to save the values.

## Property Tab

1. Click to display the Properties view, from which you can add a property to the form.

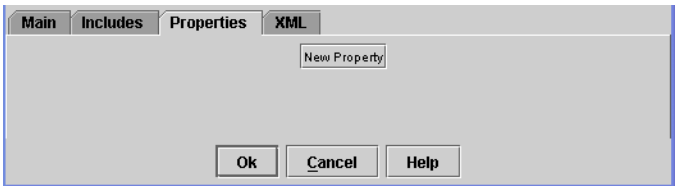



Figure 5. Properties Tab View of Form

2. To create a display property, click **New**, then double-click the new, blank entry in the Properties area to display the Property dialog.

## Includes Tab

Click the Includes tab to list the names and types of files, form fields in external forms, or view attributes to be included in the selected form. The referenced form must exist in the Identity Manager repository when this form is loaded into Identity Manager.

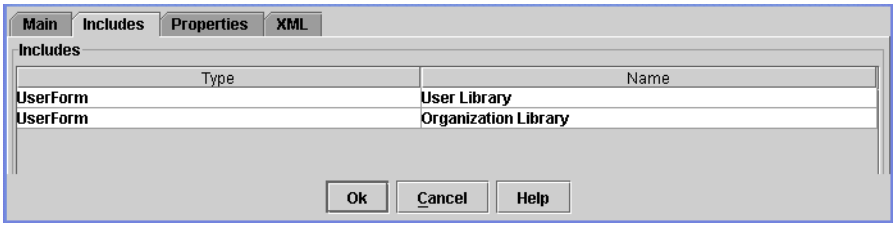



**Figure 6. Includes Tab View of Form**

This view displays the following fields:

**Type** – UserForm

**Name** – Lists full path of the file or view attribute being included.

## XML View

Click the XML tab to display a formatted XML version of the form.

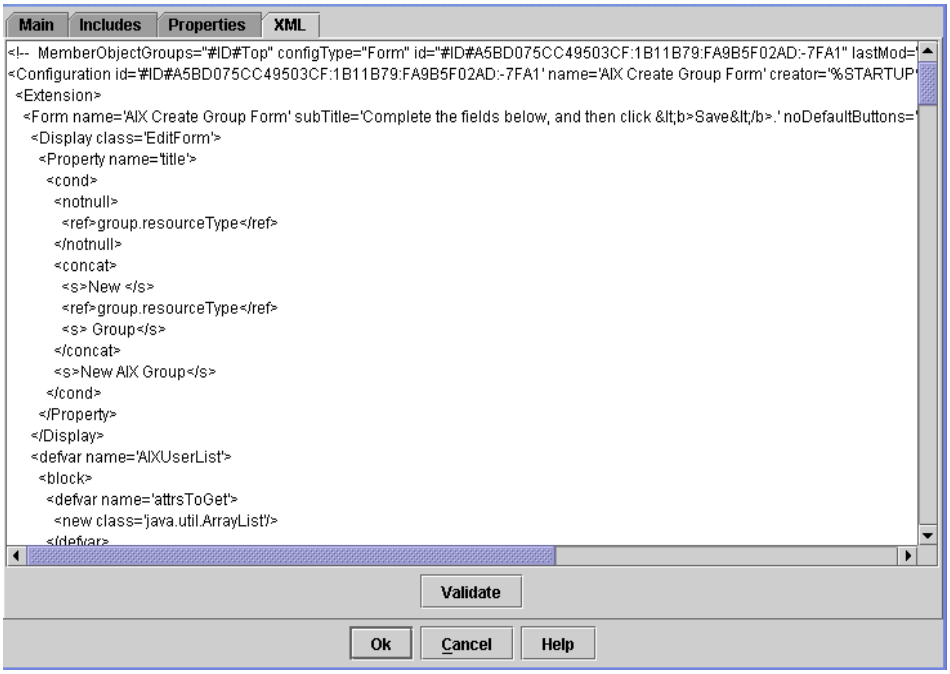



Figure 7. XML Tab View

## Creating a Field

1. In the Properties view, right-click, then select **New → Field** from the actions menu. The BPE displays a new, unlabeled field icon in the tree view of the form.
2. Double-click the new field icon to display the Field dialog. Use this dialog to set field characteristics.

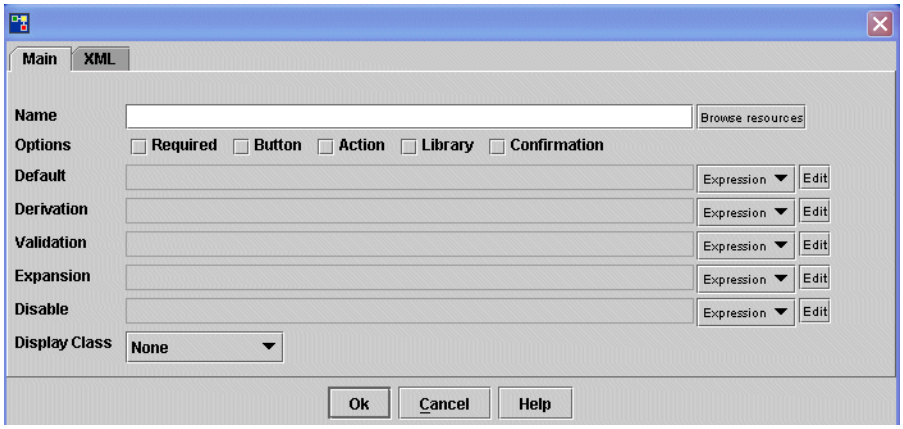



Figure 8. New Field Dialog

3. Enter the values you want, then click **OK.**

Required attributes include:

- Title
- Name
- Help Key
- Display Class

For a description of these elements, see *Form Element Dialog* later in this chapter.

**Tip** A *field name* is often a path expression into the view that is being used with this form, and is typically associated with a particular attribute on a resource. To browse a list of resources and their attributes, click **Browse resources**. The Browse resource dialog opens, displaying an expandable tree of resource types. Click the name of the resource type to display a list of resource instances and the names of their attributes. To use the name of resource attribute as your new form field name, click the resource attribute name, then click **OK**. This inserts the attribute name into the Name field.

## Creating a Loop

1. In the Properties view, right-click, then select **New → Loop** from the actions menu. The BPE displays a new, unlabeled field icon in the tree view of the form.
2. Double-click the new field icon to display the Field dialog, which you can use to set field characteristics.

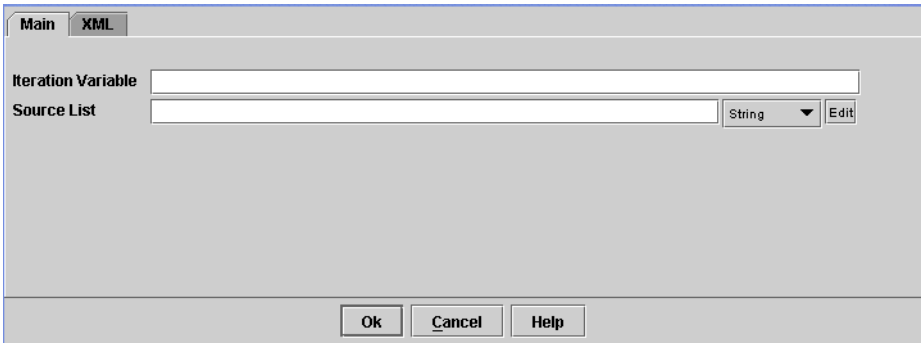



Figure 9. New Loop Dialog

3. Complete these fields, then click **OK**.

Required values:

**Iteration Variable** – Variable that defines the element over which you want to iterate.

**Source List** – Data type of the source list. Valid types include String, List, Expression, and Rule).

## Creating a Reference to Another Form or Field

1. In the Properties view, right-click, then select **New → Form Reference** or **Field Reference** from the action menu. The BPE displays a new, unlabeled form or field icon in the tree view of the form.
2. Double-click the new icon to display a dialog that you can use to set field characteristics.

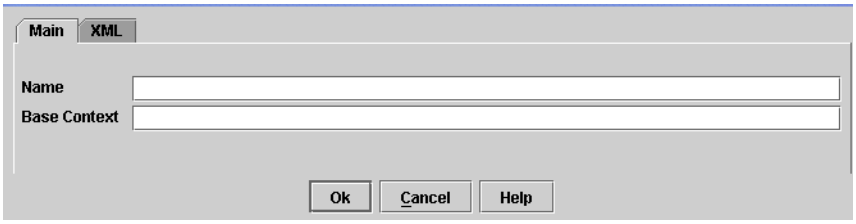



Figure 10. New Reference to Form/Field Dialog

3. Enter the following values, then click **Ok**. BPE inserts the reference in the form code with a `<FormRef>` or `<FieldRef>` element.

**Name** -- Identifies the name for this form field, typically a path expression in to the view that is used with this form.

**Base Context** -- Path used when building dynamic lists of DN strings.*Base context* identifies the underlying Map (specifically, `com.waveset.object.GenericObject` and is typically named `user` or `userview`. In the administrative interface, the editing context is user, so the base context reference is left blank. In forms launched from manual actions, such as approvals, the workflow context is the context of the form.

## Creating an Expression

**Note** See *XPRESS Language* for information on using the XPRESS language.

1. In the Properties view, right-click to select **New → Expression** from the actions menu. The BPE displays a new, unlabeled field icon in the tree view of the form.
2. Double-click the new expression icon to display the Expression popup, which you can use to create an expression.

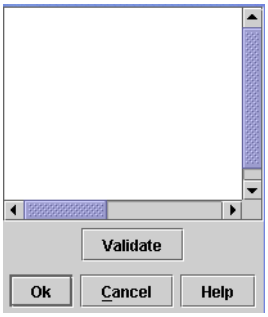



Figure 11. Expression Popup

3. In the Expression popup, right-click to access an actions menu, from which you can choose **New** →*option*, where *option* represents one of the following types of expressions:
   - Values
   - Logical
   - Strings
   - Variables
   - Math
   - Control
   - Rule

   See *XPRESS Language* for more information about these functional categories of expressions.

You can also use this menu to delete code, clear the Expression window, and change the popup display.

## Adding a Reference to an External Form Field

To include a reference to a form field in another form

1. Click the Includes tab. BPE displays the Includes view, which lists the names and types of all fields and files included in the selected form.
2. Click the **New** button. The Reference dialog opens.

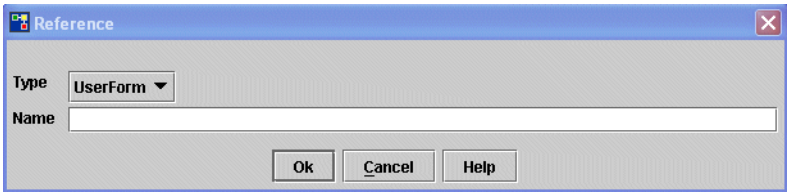



Figure 12. Reference Dialog

3. From the drop-down options menu, select the form that contains the field you want to include.
4. In the **Name** field, enter the name of the field you want to include. This creates a `<FieldRef>` element in the form.
5. Click **OK**.

## Adding a Property

1. Click the **New Property** button to display a popup of the properties available for this component type.

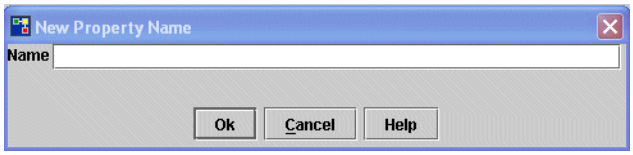



Figure 13. Available Properties Popup

OR

Select the Properties view by clicking the Properties tab. From this view, click the **New Property** button. The New Property Name dialog opens, which you can use to enter the name of the new property.


Figure 14. New Property Name popup

2. Enter the property name, then click **Ok**.

## Changing Display Class

Use the options menu to select a new HTML component display class to render this form element. Note: Refer to Display Class Reference for brief descriptions of each display class. Options include the components in the following table:

| HTML Component | Purpose |
|---|---|
| Applet | Inserts an applet reference into the page. |
| BackLink | Displays a link that returns to the previous page. |
| BorderedPanel | A container that organizes its components into 5 regions: north, south, east, west, and center. |
| Button | Displays a button. |

| HTML Component | Purpose |
|---|---|
| ButtonRow | A container that arranges its components in a horizontal row with padding in between. Typically used to display a row of Button components |
| CheckBox | Arranges its components in a horizontal row with padding in between. Typically used to display a row of Button components. (Container) |
| DatePicker | Displays a calendar icon on the page. The user can click this icon to select a calendar date and populate a page field. |
| EditForm | The default container for forms. Displays component titles in one column and components in another. Each row has an alternating gray or white background. |
| FileUpload | Variant of the Text component used for specifying the name of a file to be uploaded. |
| Hidden | A component used to include data into the HTML page that is not displayed |
| Html | Inserts pre-formatted HTML into the page. |
| Javascript | Defines JavaScript functions. |
| Label | Displays read-only text. |
| Link | Places a link on the page. |
| LinkForm | Places components in a bulleted vertical list with no titles. Typically used for pages that contain lists of Link components. Alternative to EditForm container. (Container) |
| MultiSelect | Displays a multiselection box, which displays as a two-part object in which a defined set of values in one box can be moved to a “selected” box. |
| NameValueTable | Displays a list of name/value pairs in a simple table with a beige background. |
| Panel | Organizes its components in either a horizontal or vertical line.(Container) |
| Radio | Displays a horizontal list of one or more radio buttons. A user can select only one radio button at a time. If the component value is null or does not match any of the allowed values, no button is selected. |
| SectionHead | Displays a section heading. These are recognized by the EditForm container to and are rendered in bold text that spans both the title and component columns. |

| HTML Component | Purpose |
|---|---|
| Select | Displays a single-selection list box. |
| SimpleTable | Arranges components in a simple grid with a row of column titles. |
| SubTitle | Identifies the text that displays below the form title. |
| Text | Default display type. Displays a regular text entry box. |
| TextArea | Displays a multi-line text entry box. |
| Title | Identifies the text that displays at the top of the form. |

## Form Element Dialog

Double-clicking a form element name in the tree view opens the Form Element dialog. From this dialog, you can view or change the following types of information about the element.

This dialog provides two views of the element information:

- **Main** tab view – Similar in style to the Main view of the Form dialog.
- **XML** tab view – Displays the XML for the element.

## Main View

Click the Main tab on the Element dialog to display this view (typically the default display).

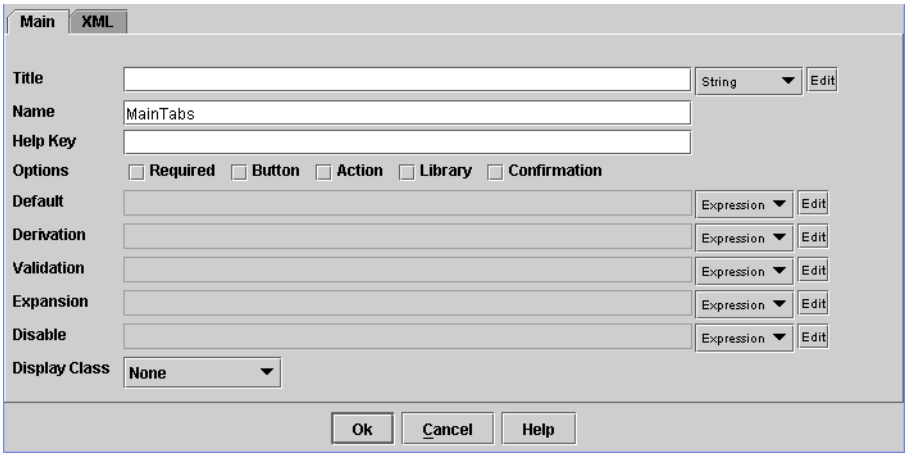



Figure 15. Form Element Dialog (Main View)

This dialog closely resembles the Form dialog. Not all fields in the following table are displayed for each selected element. For example, the `maxLength` property is displayed for text elements only.

| Field | Description |
|---|---|
| Name | Enter the name for this field. A field name is typically a path expression into the view that is being used with this form. All fields that display as editing components (such as text boxes, checkboxes, and selects) must have a name that specifies a view path. Fields that do not display as editing components (such as `SectionHead` and `Javascript`) do not require names. However, you can give non-editing fields names if they need to be referenced by another form through a Field reference. |
| Title | Enter a title for the field. This title displays adjacent to the field on the form. Select the data type of this element from the drop-down menu immediately adjacent to this field. To edit the text displayed in this field, click the adjacent **Edit** button. |
| Help Key | Enter the help key that associates guidance help with the field. This value is the name of an entry in an associated help catalog specified by the form. Specifying a help key causes an icon to appear to the left of the field. Moving the mouse over the icon causes the text referenced in the help catalog to display. |

| Field | Description |
|---|---|
| Options | Select one or more display options for the field:<br>• **Required** -- An entry or selection in this field is required to process the form.<br>• **Button** -- Causes the field to display in a single, horizontal row at the bottom of the form. Otherwise, it displays on the next line of the form. This is most commonly set with fields that use the display class `Button`.<br>• **Action** -- When set, a change causes the page to refresh any `Select` or `MultiSelect` controls. In the Identity Manager Administrator Interface, this causes the underlying view to be refreshed. Role selection exemplifies this behavior. When a new role is selected in the Tabbed User Form, the view is refreshed to reflect the resources that are assigned through that role during that edit session. After the view has been refreshed, resource account attributes on those newly assigned resources can be explicitly set.<br>• **Library** -- Indicates that a field should only display when it is referenced, rather than when it is declared. This is useful when the order in which fields are evaluated on a form may differ from the order in which they are displayed to the user. |
| Default | Specify an expression to calculate a default value for the field. The default expression is called before the form is displayed if the current value for this field is null. |
| Derivation | Specify an expression to calculate the value of a field before it is displayed. It is similar to a Default expression, except that it is evaluated even if the current field value is non-null. The derivation expression is evaluated before the form is first displayed, and then again each time the form is refreshed. |
| Validation | Specify logic to determine whether a value entered in a form is valid. Validation expressions return null to indicate success, or a string containing a readable error message to indicate failure. Validation rules are evaluated only when a form is submitted, not after each refresh or recalculate. |
| Expansion | Specify an expression to calculate the value of the field after the form has been submitted. Expansion expressions are typically used with fields that are also marked hidden. Since hidden fields are not directly editable by the user, the value can be calculated with an Expansion expression. |
| Disable | Specify an expression that, if evaluated to true, disables the field and any of its nested fields. A disabled field does not display on the form. It is commonly used to determine if a user has a specific type of resource. If the user does, the form then displays the appropriate fields for that resource. |

| Field | Description |
|---|---|
| Display Class | Identifies the HTML component class used to render this form component in the browser. By default, the Display Class selection is EditForm. If the form is a link form (such as the End User menu), then select LinkForm from the Display Class options.<br>See the HTML Display Class table in *HTML Display Components*. |
| value | Specifies the property attribute. Typically is a String. |
| maxLength | Specifies the maximum number of characters for this element. |

From this tab, you can change the data type of the following form elements:

- Default
- Derivation
- Validation
- Expansion
- Disable

Drop-down options menus adjacent to the field displays the valid data type options.

## XML View

Click the XML tab to display the XML for the selected form element.

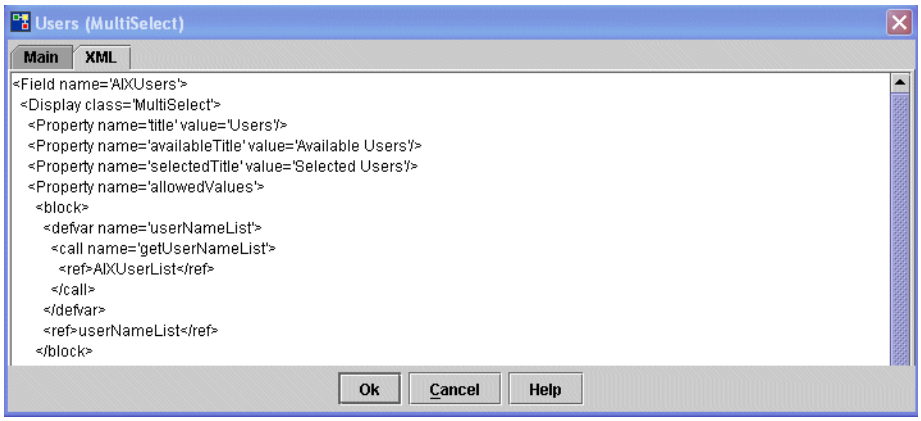



Figure 16. XML View of MultiSelect Element

# About the Forms Toolbox

Use the toolbox to quickly drag and drop form elements into the BPE property view.

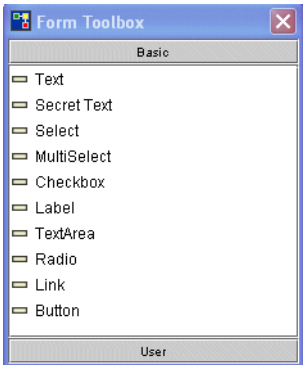



Figure 17. Forms Toolbox Dialog

To open the toolbox, right-click in the diagram view and select the toolbox option.

## Default Services

Toolbox elements are divided into two categories: Basic and user.

| Service | Description |
|---|---|
| Text | Displays a regular text entry box. |
| Secret Text | Displays text as asterisks (*). Typically used for encrypted data like passwords |
| Select | Displays a single-selection list box. Values for the list box must be supplied by the `allowedValues` property. |
| MultiSelect | Displays a multiselection text box, which displays as a two-part object in which a defined set of values in one box can be moved to a selected box. Values in the left box are defined by the `allowedValues` property, values are often obtained dynamically by calling a Java method such as `FormUtil.getResources`. The values displayed in the right side of a multiselection box are populated from the current value of the associated view attribute, which is identified through the field name. |
| Checkbox | Displays a checkbox. When checked, the box represents a value of `true`. An unselected box represents a `false` value. |

| Service | Description |
|---|---|
| Label | Displays a string of text. |
| TextArea | Displays a multi-line text entry box |
| Radio | Displays a horizontal list of one or more radio buttons. A user can select only one radio button at a time. If the component value is null or does not match any of the allowed values, no button is selected. |
| Link | Places a link on the page. |
| Button | Displays a button. |
| accountId | Refers to a variable that is defined by the view that is used with this form. |

# Additional Customization-Related Topics

The following topics are covered in this section:

- Form structure
- Form components
- Defining fields
- Guidelines for structuring a form

## Form Structure

Forms are stored as XML objects within the Identity Manager repository. Each form is stored as its own object with the following structure.

**Note** You do not need to know the XML structure of a form. The Business Process editor simplifies working with form structure. This information is supplied for your reference only.

The following stub form illustrates the general structure of a form.

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE Configuration PUBLIC 'waveset.dtd' 'waveset.dtd'>
<!--  id="#ID#UserForm:EndUserMenu" name="End User Menu"-->
 <Configuration id='#ID#UserForm:EndUserMenu' name='End User Menu'
createDate='1012185191296' lastModifier='Configurator'
lastModDate='1013190499093' lastMod='44' counter='0'
wstype='UserForm'>
   <Extension>
```

```
   <Form name='End User Menu'>
      <Display class='LinkForm'>
         <Property name='title' value='User Self Service'/>
         <Property name='subtitle' value='Select one of the following
         options'/>
      </Display>

… Field content

   </Form>
   </Extension>
<MemberObjectGroups>
   <ObjectRef type='ObjectGroup' id='#ID#Top' name='Top'/>
   </MemberObjectGroups>
</Configuration>
```

## Form Components

The following table identifies form components in the order in which they appear in the form. Each form component is discussed in greater detail below.

| Form Component | Purpose |
|---|---|
| header | Introduces information about the form object definition. Includes start tags for `<Form>`, `<Extension>`, and `<Configuration>` elements and defines form properties (such as `title`, `subtitle`, `titleWidth` displayed when the form is launched). |
| form body | Contains field definitions, form functions, form variables. This is the part of the form that you will edit. |
| footer | Closing tags for `<Form>`, `<Extension>`, and `<Configuration>` elements. |

## Header

The form header includes:

- Standard introductory information included in XML files: the XML declaration and documentation declaration, including the DTD associated with this XML file. In the preceding example, this introductory information is:

  ```
  <?xml version='1.0' encoding='UTF-8'?>
  <!DOCTYPE Configuration PUBLIC 'waveset.dtd'
  'waveset.dtd'>
  ```

  This system appends this information to the file. Do not edit.
- Start tags for the `<Extension>` and `<Configuration>` elements, which surrounds the HTML components that describe the form's appearance and behavior. The `Configuration` element contains attributes that describe the form object properties.

The header contains information about the form, including internal identification such as date of creation, login of whoever last modified the file, and the form type. The page processor typically generates this information.

**Note: The system generates the following information for internal use only. Do not edit these attributes.**

| Element | Definition | Syntax/Example |
|---|---|---|
| `<Extension>` | Required to wrap the `<Form>` element. | `<Element>…</Element>` |
| `<Configuration>` | Contains information that the system uses internally when processing the form object, including the date of last modification and login of the user who last modified this form. Most of this information is typically associated with any persistent object that is stored in the Identity Manager repository. You typically do not need to edit this information. | `Configuration id='#ID#UserForm:EndUserMenu' name='End User Menu' createDate='1012185191296' lastModifier='Configurator' lastModDate='1013190499093' lastMod='44' counter='0' wstype='UserForm'>` |

## Form Body

The *form body* is composed of:

- Form properties, which include `title`, `subtitle`, and `width`. These properties are defined in the table titled Form Properties.
- `Field` elements, which you use to determine the appearance and function of the fields as they appear to the user in the product interface. Fields can also contain XPRESS logic to calculate information. For more information on using the XPRESS language, refer to *XPRESS Language*.

The following table lists form header properties.

| Property | Purpose |
|---|---|
| `title` | Identifies the text that appears at the top of the form. Typically, this title is in a bold font typically larger than the other font on the screen. The form `title` appears under the Identity Manager page. You cannot edit the display characteristics of title.<br>In the example given in the section titled *Form Components*, the value of `title` is `User Self Service` |
| `subtitle` | Identifies text that appears under title of the form on the page defined by this form. You cannot edit the display characteristics of title.<br>In the preceding example, the value of `subtitle` is `Select one of the following options` |
| `titleWidth` | Defines the width in pixels of the value of title in the browser window.<br>**Example**<br>`<Display>`<br>`  <Property name='titleWidth'>`<br>`  <Integer>120</Integer>`<br>`  </Property>`<br>`</Display>` |

The following table lists all elements that can occur within the form body.

| Component | Definition | Example |
|---|---|---|
| defun | Defines an XPRESS function. This element can be called by any field element in a form. | <defun name='add100'><br><def arg name='x'/><br><add>x100<br></add><br></defun> |
| defvar | Defines an XPRESS variable that is commonly used to hold the results of a computation. | <defvar name='nameLength'<br><length><br><ref>fullname</ref><br></length><br></defvar> |
| Display | Identifies the display components that will define the appearance of the field. See the section titled *Display Element* for more information. | <Display class='LinkForm'><br><Property name='title' value='User Self Service'/><br><Property name='subtitle' value='Select one of the following options'/><br></Display> |
| Field | Main element used within the form body. See the section titled *Field Element* for more information. | <Field name='fullname'/> |
| FieldRef | Provides a reference to a field defined in an included form. | <FieldRef name='fieldName'/> |

| Component | Definition | Example |
|---|---|---|
| Include | Provides a reference to another form object. Once included in the current form, the fields defined in the form can be referenced and displayed. | <Include><br><ObjectRef type='UserForm' id='#ID#UserForm:UserForm Library'/><br></Include> |
| FormRef | Provides a reference to another form object. | <FormRef name='formName'/> |
| Namespace | Provides a way to define a shortcut to a view. The shortened name can then be used in field names and references instead of the longer name. When using the name substitution, use a colon (:) following the name. | <Namespace name='w' value='waveset'/> |

### Form Element

The <Form> element must surround all Field elements and contains the unique name of the form. The elements listed on the previous page are contained within the beginning and ending Form tags.

```
<Form name='Create User Form'
   <Field name='waveset.accountId'>
             … additional fields
</Form>
```

Additional example:

```
<Form name='Task Launch Form'>
   <Display class='EditForm'>
      <Property name='title' value='Task Launch'/>
      <Property name='subTitle' value='Enter task launch parameters'/>
   </Display>
                ...
</Form>
```

### Display Element

A Display element within the Form element describes the component that will be used to render the form. By default, this Display element is the commonly used EditForm component. You will rarely need to change the Form component class, but you can set component properties. The two most common properties to specify are title and subTitle.

`EditForm` also supports the `adjacentTitleWidth` property, which can be used to set the width of the titles of adjacent fields. If this property is not defined, it defaults to zero.

If you define `adjacentTitleWidth` as equal to zero, columns titles will automatically resize. If set to a non-zero value, then the title width of adjacent columns (for example, the second and third columns) will be the value of `adjacentTitleWidth`.

```
<Form name='Default User Form' help='account/modify-help.xml'>
   <Display class='EditForm'>
     <Property name='titleWidth' value='120'>
     <Property name='adjacentTitleWidth' value='60'>
   </Display>
```

### Field Element

The `Field` element is the main element used within the form body. Fields are used to define each of the user's attributes. You can use `Field` elements to include XPRESS logic in form fields. For more information on working with form field elements, refer to the section titled *Defining Fields*.

The following example creates an editing field with the label Email address.

```
<Field name='waveset.email'>
      <Display class='Text'>
         <Property title='Email Address'/>
         <Property size='60'/>
         <Property maxLength='128'/>
      </Display>
   ...
</Field>
```

The name of an editing field is typically a path expression within a view that is being used with the form. In this example, `waveset.email` refers to the email address associated with a user object in the Identity Manager repository.

## Footer

The footer contains information about the Identity Manager object group or organization with which the form is associated. It also contains the closing tags for the `</Form>`, `</Extension>`, and `</Configuration>` elements or other elements opened in the header. The footer in the preceding example is:

```
</Form>
</Extension>
   <MemberObjectGroups>
      <ObjectRef type='ObjectGroup' id='#ID#Top' name='Top'/>
   </MemberObjectGroups>
```

```
</Configuration>
```

<MemberObjectGroups> identifies the object group or organization into which the system stores an object. If you do not specify an object group, by default the system assigns the object to the Top organization. For Configuration objects that contain forms, are typically found in the All group with this syntax:

```
<MemberObjectGroups>
   <ObjectRef type='ObjectGroup' name='All'/>
</MemberObjectGroups>
```

## What Is a Form Field?

The form body contains Field elements that define how each element of the Web page appears and behaves. Each Field can contain other fields, each with its own display component.

Form fields comprise several parts, which are encapsulated by the <Field> tag set:

- **Value Expressions**. The field can contain a number of XPRESS expressions, which calculate the value of the field or define the set of allowed values. For example, <Default> is used to define the default value of a field, and <Derivation> is used to derive the value for the field when the form is first loaded. Not all field elements contain expressions. See the section titled *Defining Field Names*.
- **HTML Display Components**. Display components determine how visible elements are displayed. In Identity Manager form fields, *display components* (defined in the form by the <Display> element) determine the behavior and appearance of form fields. You can specify only one display component for each field. These display components are described in detail in the section titled *HTML Display Components*.
- **Disable Expressions**. Fields can be conditionally included in the form by using Disable expressions. If the Disable expression evaluates to true, the field is ignored.

In the BPE Tree view, form fields are displayed with the field icon 

## Creating Variables

Use the following syntax to include variables that contain long lists of constant or static data. This syntax builds a static list once and reuses it on each reference

```
<defvar name='states'>
    <List>
      <String>Alabama</String>
            ...
    </List>
</defvar>
```

The former syntax is preferable to `<list><s>Alabama</s>...</list>`, which builds a new list each time it is referenced.

## Defining Fields

This section describes procedures you perform when customizing any form. These procedures include:

- Defining field names
- Defining field elements
- Adding a visible field
- Hiding a field. When you hide a field, the field (and any fields nested within it) is not displayed on the page, but its value is included in the form processing.
- Disabling a field. When you disable a field, the field (and any fields nested within it) is not displayed in the page, and its value expressions are not evaluated. If the view already contains a value for the disabled field, the value will not be modified.
- Setting a field value
- Calling functions

### Using the BPE to Define a New Field

From the BPE, you can add field to a form by:

1. Right-click in the Property view in the Main display of the form to access the actions menu.
2. Select New -> Field. BPE displays a field icon below the last field icon in the expanded Tree view of the form. No field name follows the icon.
3. Click the new field icon to display a Form Field dialog. Define the field characteristics.
4. When you've finished setting field characteristics, click **OK**.

The following sections discuss in more detail the field characteristics you will set.

## Defining Field Names

You use the field name to match the attribute defined on the resource to the text entry field that is displayed on the web page. When the resource is defined, the system sets up a schema map that maps resource account attributes to Identity Manager attributes. For example, your Active Directory resource might have attributes that include `firstname, lastname`, and `Office Phone.` When referring to these attributes in the form, you must know the name of the attribute on the Identity Manager schema plus the path to the attribute from the view.

There are two ways of defining the `name` attribute of the `Field` element:

- The `name` attribute typically contains a path to an attribute within the user view.
- The `name` attribute is used to identify the field so that it can be referenced by other fields in the form or by a `FieldRef` element. This occurs when fields are defined to represent containers of other fields and do not correspond to any one attribute of the view.

Determining whether a `Field` name represents a path expression for the view or is simply a reference name depends on the value of the `class` attribute selected in the `Display` element. If the display class is the name of an editing component class, then the name is expected to be a path expression for the view. See the section titled *HTML Component* in the *HTML Display Components* chapter for a detailed explanation of the component classes.

### Creating a Path Expression to a View Attribute

Typically, you define a `Field` name by including the path to an attribute in the user view (the path expression). For a list of these attributes, see *Views*.

#### Example

The following field definition renders a text field to edit the Identity Manager email address:

```
<Field name='waveset.email'>
   <Display class='Text'>
      <Property name='size' value='60'/>
   </Display>
</Field>
```

The string `waveset.email` is a path expression for the user view that targets the email address stored in the Identity Manager repository.

### Example

This example field edits the email address defined for a particular resource account. The field name references a resource in the account:

```
<Field name='accounts[Microsoft Exchange].email'>
   <Display class='Text'>
      <Property name='size' value='60'/>
   </Display>
</Field>
```

The string `accounts[Microsoft Exchange].email` is a path expression to another location within the user view that holds information about account attributes for a specific resource. In this example, the resource named `Microsoft Exchange`.

### Example

This example field defines the email address for all resources including Identity Manager that contain an attribute named `email` on the left side of the schema map.

```
<Field name='global.email'>
   <Display class='Text'>
      <Property name='size' value='60'/>
   </Display>
</Field>
```

## Identifying the Field for Reference

Naming a field provides you a way to reference the field value in other fields. Use the `<ref></ref>` tag set to reference a field value from another field. The following example concatenates into the `fullname` field the `firstname` and `lastname` field values with a string, a comma, and a space such as: `lastname, firstname`. The `<s>` tag designates a string.

### Example

```
<Field name='global.firstname'>
   <Display class='Text'/>
</Field>
<Field name='global.lastname'>
   <Display class='Text'/>
</Field>
<Field name='global.fullname'>
   <Expansion>
      <concat>
         <ref>global.lastname</ref><s>, </s>
         <ref>global.firstname</ref>
      </concat>
   </Expansion>
</Field>
```

Not all `Field` names represent path expressions for the view. Some fields are defined to represent containers of other fields and do not correspond to any one attribute of the view. In these cases, the `Field` name is used to identify the field so that it can be referenced by a `FieldRef` element. If the field does not need to be referenced, you do not need to specify the name.

For example, a form button performs an action, but does not contain a value or need to be referenced by another form. Therefore, it does not need a field name:

```
<Field>
   <Display class='Button'>
      <Property name='label' value='Recalculate'/>
      <Property name='command' value='Recalculate'/>
   </Display>
</Field>
```

For more information on user views, see the section titled *User Views and Forms*.

## Field Display Properties

The `Display` element is common to all visible form fields. `Display` elements contain `Property` elements that define the characteristics of the field rendered by the browser. By defining a `Display` element for a form, it will be visible on the screen unless there is a `Disable` element in the field that evaluates to true. There can be conditions in which the form is displayed until another field or value is set and when the form recalculates the field can become hidden from the screen. See the section titled *Disabling Fields*.

### Defining Element Display Type in the BPE

1. In the Form Element dialog, click on the Display Class dropdown menu icon. BPE displays a menu of the available display class types.

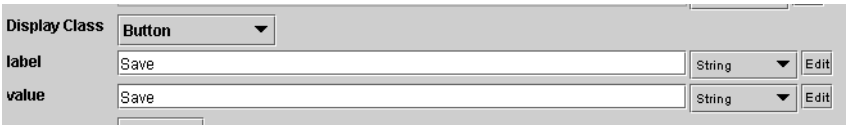



Figure 18. Display Class Area of Form Element Dialog

2. After you choose the display class type, the dialog changes and shows attributes relevant to the display type. For example, if you select Text from the Display Type pull-down menu, the dialog changes to display the `size` and `maxLength` fields.

`Display`

Describes class and properties of the visible field. This element specifies a component class to instantiate and a set of property values to assign to the instance.

### Example

```
<Display class='Text'>
   <Property name='size' value='20'/>
   <Property name='maxLength' value='100'/>
</Display>
```

The `class` attribute of the `Display` element must be the name of a `Component` class. By default, these classes are expected to reside in the `com.waveset.ui.util.html` package and include Applet, Button, and DatePicker among others. A list of all the default classes and their descriptions can be found in the *Base Component Class* section of the *HTML Display Components* chapter. To reference a class that is not in this package, you must use a fully qualified class name for the `class` attribute. All classes described in this document are in the default package and do not require qualified names.

`Property`

Occurs within the `Display` element. These define the names and values of properties that are to be assigned to the component. The property name is always specified with the `name` attribute.

## Specifying Property Values for a Display Element

You can specify the `Property` value for a `Display` element through the use of:

- a `value` attribute
- an XML Object language
- an expression to specify a value

For most property values, you can use the `value` attribute and let the system coerce the value to the appropriate type.

### Use of the value Attribute

The most common way of specifying the property value is with the `value` attribute. The value of the `value` attribute is treated as a string, but if necessary, the system will coerce it to the data type desired by the component. (See the section titled *Data Types* for more information on type coercion.) In the previous example, the property `size` is set to the integer value 20, and the property `maxLength` is set to the integer value 100.

### Example of Display Element in a Field Component

The following example creates a field that uses `SimpleTable` to organize several subfields. Within XML forms, the most common `Container` components used are `SimpleTable` and `ButtonRow`.

```
<Field name='SelectionTable'>
   <Display class='SimpleTable'>
      <Property name='columns'>
         <List>
            <String>Account</String>
            <String>Description</String>
         </List>
      </Property>
   </Display>

<Field name='accounts[LDAP].selected'>
   <Display class='Checkbox'>
      <Property name='label' value='LDAP'/>
   </Display>
</Field>

<Field>
   <Display class='Label'>
      <Property name='text' value='Primary Corporate LDAP Server'/>
   </Display>
</Field>

<Field name='accounts[W2K].selected'>
   <Display class='Checkbox'>
      <Property name='label' value='Windows 2000'/>
   </Display>
</Field>

<Field>
   <Display class='Label'>
      <Property name='text' value='Primary Windows 2000
             Server'/>
   </Display>
</Field>

</Field>
```

The browser displays the preceding code as shown below.

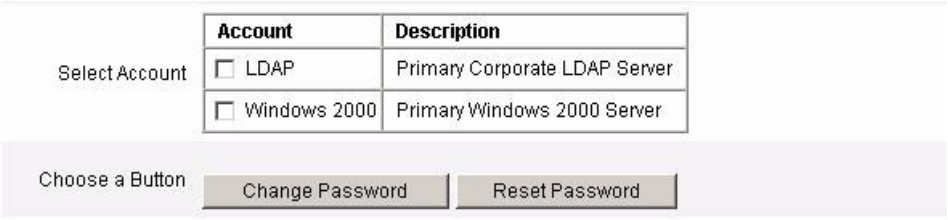



Figure 19. SimpleTable and ButtonRow Components as Displayed in the Browser

Within the `Display` element are zero or more `Property` elements. These define the names and values of properties that are assigned to the component. The `Property` name is always specified with the `name` attribute. The property value is most commonly specified with the `value` attribute. The value of the `value` attribute treated as a string, but if necessary it will be coerced to the data type desired by the component.

### Use of XML Object Language

You can also specify property values using the XML Objects language. This approach is useful primarily when specifying list values. This language provides a syntax for describing several standard Java objects as well as other objects defined by Identity Manager.

The more common Java XML objects include:

- `List`
- `Map`
- `MapEntry`
- `String`
- `Integer`
- `Boolean`
- `Object`

When you use the XML Object syntax to specify property values, an element is placed inside the `Property` element. For more information on the XML Object language, see *XML Object Language*.

### Example

```
<Property name='size'>
   <Integer>10</Integer>
</Property>

<Property name='title'>
   <String>New Password</String>
</Property>

<Property name='leftLabel'>
   <Boolean>true</Boolean>
</Property>

<Property name='allowedValues'>
   <List>
      <String>Texas</String>
      <String>Iowa</String>
      <String>Berkshire</String>
   </List>
</Property>
```

All properties that expect list values recognize the `List` element. Most attributes, in addition, recognize the comma list syntax for specifying lists.

### Use of an Expression to Calculate the Value

You can also specify a `Property` value through an expression. This allows a value to be calculated at runtime, possibly combining fixed literal values with variable values defined by the page processor.

### Example

```
<Property name='title'>
   <concat>
      <s>Welcome </s>
      <ref>waveset.accountId</ref>
      <s>, select one of the following options.</s>
   </concat>
</Property>
```

In the preceding example, `waveset.accountId` is a reference to a variable. When the system generates the HTML for this component, the page processing system supplies the value for the `waveset.accountId` variable. The names of the variables that can be referenced are defined by the page processor. In most cases, these are defined by a *view* that is used with the XML form. Form designers must be aware of the view with which the form will be used and only reference attributes defined by that view.

## Defining Field Elements

To create any type of field element using the BPE

1. Right-click in the Property pane, then select **New → Field**. The tree view displays a new Field icon.
2. Double-click the new Field icon to display the Form Element dialog for the new field.
3. In the Form Element dialog, select the display class that will define the new field from the Display Class menu.

### Disable Element

Calculates a Boolean value. If true, the field and all its nested fields will be ignored during current form processing.

**Example**

This example illustrates a field definition that uses an expression within the `<Disable>` element to control the visibility of the field. `accountInfo.typeNames` is used to find the type of all resources that a user is assigned to. The type returned is a list of all the user's resource types (for example, `Windows NT)`. If the list of returned type names contains Solaris, then this field is displayed on the screen. Otherwise, this field is disabled.

```
<Field name='HomeDirectory' prompt='Home Directory'>
   <Display class='Text'/>
      <Disable>
         <not>
            <contains>
               <ref>accountInfo.typeNames</ref>
               <s>Solaris</s>
            </contains>
          </not>
   </Disable>
</Field>
```

From the BPE, you set a `Disable` element from the Form element dialog for the selected element.

## Default Element

Calculates a value to be used as the value of this field, but only if the field does not already have a non-null value. `Default` is essentially the same as `Derivation`, except that the value applies only if the current value is non-null. Default expressions are calculated when:

- the form is first loaded
- data is retrieved from one or more resources
- the form is recalculated or saved until the field value is non-null.

**Example**

This example shows a field definition that uses string manipulation expressions to return a default account ID composed of the first initial of the first name plus the user's last name.

```
<Field name='waveset.accountId'>
   <Display class='Text'>
      <Property name='title' value='AccountID'/>
   </Display>
   <Default>
      <concat>
         <substr>
            <ref>accounts[Exchange].firstname</ref>
               <i>0</i>
               <i>1</i>
            <ref>accounts[Exchange].lastname</ref>
         </substr>
      </concat>
   </Default>
</Field>
```

From the BPE, you set a `Default` element from the Form element dialog for the selected element.

## Derivation Element

Unconditionally calculates a value for the field. Whenever a Derivation expression is evaluated, the current field value is replaced.

`Derivation` expressions are calculated when the form is first loaded or data is returned from one or more resources

### Example

The following example shows a field definition that uses *conditional logic* to map one set of values into another set. When this field is processed, the expression in the `<Derivation>` element is evaluated to determine the descriptive value to be displayed for this field based on the location code returned from the resource.

```
<Field name='location'>
   <Display class='Text'>
      <Property name='title' value='Location'/>
   </Display>
   <Derivation>
         <switch>
            <ref>accounts[Oracle].locCode</ref>
            <case>
               <s>AUS</s>
               <s>Austin</s>
            </case>
            <case>
               <s>HOU</s>
               <s>Houston</s>
            </case>
            <case>
               <s>DAL</s>
               <s>Dallas</s>
            </case>
            <case default='true'>
               <s>unknown</s>
            </case>
         </switch>
   </Derivation>
</Field>
```

From the BPE, you set a `Derivation` element from the Form element dialog for the selected element.

## Expansion Element

Unconditionally calculates a value for the field. It differs from Derivation in the time at which the expression is evaluated.

Expansion statements are calculated when:

- the page is recalculated
- the form is saved

### Example

The following example shows a field definition that uses conditional logic to convert the value derived for the `location` field in the previous example back into a three-letter abbreviation that will be stored on the Oracle resource. Notice the difference in the field names. The location field value is not saved on any resource. It is used to calculate another field.

```
<Field name='accounts[Oracle].locCode'>
   <Expansion>
      <switch>
         <ref>location</ref>
         <case>
            <s>Austin</s>
            <s>AUS</s>
         </case>
         <case>
            <s>Houston</s>
            <s>HOU</s>
         </case>
         <case>
            <s>Dallas</s>
            <s>DAL</s>
         </case>
      </switch>
   </Expansion>
</Field>
```

From the BPE, you set an `Expansion` element from the Form element dialog for the selected element.

## Validation Element

Determines whether a value entered in a form is valid. Validation rules are evaluated whenever the form is submitted.

### Example

This example Validation rule checks to make sure that a user's zip code is five digits.

```
<Validation>
   <cond>
      <and>
      <eq><length><ref>global.zipcode</ref></length>
            <i>5</i>
      </eq>
            <gt><ref>global.zipcode</ref><i>99999</i></gt>
      </and>
      <null/>
```

```
      <s>zip codes must be five digits long</s>
   </cond>
</Validation>
```

From the BPE, you set a `Validation` element from the Form element dialog for the selected element.

### Editing and Container Fields

When the `Display` element appears with the `Field` element, it describes the component that will be used to render that field. There are two types of fields:

- **editing fields**. These are associated with a particular value to modify.
- **container fields**. These surround one or more fields.

Editing fields must have names and are always used with one of the editing components such as `Text` or `Checkbox`.

**Example Editing Field**

```
<Field name='waveset.email'>
   <Display class='Text'>
      <Property title='Email Address'/>
      <Property size='60'/>
      <Property maxLength='128'/>
   </Display>
      ...
</Field>
```

The name of an editing field is typically a path expression within a view that is being used with the form. In the preceding example, `waveset.email` refers to the email address associated with a user object in the Identity Manager repository.

A `Container` field may not have a name and is always used with one of the `Container` components, such as `ButtonRow`, `SimpleTable`, or `EditForm`.

One common type of container is the `EditForm` container, which builds an HTML table that contains titles in one column and components in another. These titles are defined in the `title` property and are rendered on the Identity Manager page associated with the form.

## Disabling Fields

When you disable a field, the field (and any fields nested within it) is not displayed in the page, and its value expressions are not evaluated or incorporated in to any `global.*` attributes. If the view already contains a value for the disabled field, the value will not be modified.

```
<Disable></Disable>
```

### Example

```
<Field name='waveset id'>
   <Display class='Text'>
      <Property title='accountId'>
   </Display>
   <Disable>
      <eq><ref>userExists</ref><s>true</s></eq>
   </Disable>
</Field>
```

**Note** Disable expressions are evaluated more frequently than other types of expression. For this reason, keep any `Disable` expression relatively simple. Do not call a Java class that performs an expensive computation, such as a database lookup.

Use caution when referencing fields with `Disable` rules. Otherwise, fields inside containers might be disabled.

## Hiding Fields

When you hide a field, the field (and any fields nested within it) is not displayed on the page, but its value is included in the form processing.

To hide a field, simply do not assign a Display class to the field.

```
<Field name='field A'/>
```

You can use the BPE to hide fields in two ways:

- From the Property Sheet view of the form, toggle on the Hidden button in the Options area.

OR

- From the Form Element dialog for the element, select Hidden as the Display class.

## Calculating Field Values

Field values can be calculated from the values of other fields or any logical expression. For example, you can calculate the user's full name from the first name, middle initial and last name.

**Example**

```
<Field name='global.fullname'>
   <Expansion>
      <concat>
         <ref>global.firstname</ref>
         <s> </s>
         <ref>global.middle</ref>
         <s> </s>
         <ref>global.lastname</ref>
      </concat>
   </Expansion>
</Field>
```

## Setting Default Values

You can set the email address based on the user's first initial and the first seven characters of the user's last name. In this example, the system performs an additional check to ensure that the values have been set before performing the concatenation. This additional check is performed to:

- Allow the email address to set only when the account is first created.
- Confirm that the first and last name fields have been set.

**Example**

```
<Field name='global.email'>
   <Default>
      <and>
         <notnull><ref>global.firstname</ref></notnull>
         <notnull><ref>global.lastname</ref></notnull>
      </and>
         <concat>
         <downcase>
            <substr>
            <ref>global.firstname</ref>
            <i>0</i>
            <i>1</i>
            </substr>
         </downcase>

         <downcase>
            <substr>
            <ref>global.lastname</ref>
            <i>0</i>
            <i>6</i>
            </substr>
         </downcase>
```

```
         <s>@waveset.com</s>
      </concat>
   </Default>
</Field>
```

## Deriving Field Values

Some fields are used on the form solely to calculate other fields. These fields cannot be stored on any resource to which the user belongs. When the user record is edited, each of the resources is contacted and the field values for the attributes are populated. To populate the fields that are used for calculations, you can write derivation rules.

### Example

A phone number field can be represented on the form as a single text box. However, a more advanced form might have three fields for the area code and phone number, which are used to calculate the phone number that is saved to the resource.

In the simple case of representing a phone number, you can have form fields that resemble the ones listed below.

### Example

```
<Field name='P1'>
   <Display class='Text'>
      <Property name='title' value='Office Phone Number'/>
      <Property name='size' value='3'/>
      <Property name='maxLength' value='3'/>
   </Display>
</Field>
<Field name='P2'>
   <Display class='Text'>
      <Property name='title' value='-'/>
      <Property name='size' value='3'/>
      <Property name='maxLength' value='3'/>
   </Display>
</Field>
<Field name='P3'>
   <Display class='Text'>
      <Property name='title' value='-'/>
      <Property name='size' value='4'/>
      <Property name='maxLength' value='4'/>
   </Display>
</Field>
<Field name='global.OfficePhone'>
   <Expansion>
      <concat>
```

```
         <ref>P1</ref><s>-</s>
         <ref>P2</ref><s>-</s>
         <ref>P3</ref>
      </concat>
   </Expansion>
</Field>
```

**Example**

The following example expands on the field definition for the field P1 defined above. It defines how a phone number attribute is read into the form, and consequently expands into the three field displays.

```
<Field name='P1'>
   <Display class='Text'>
      <Property name='title' value='Office Number'/>
      <Property name='size' value='3'/>
      <Property name='maxlength' value='3'/>
   </Display>
</Field>
```

When a user enters data into Identity Manager, the form can ensure the data is entered properly. However, Identity Manager cannot ensure that data entered directly into the resource meets the same requirements. For example, over the years, administrators might have entered the phone number as 123-4567 (8 characters), 123-123-4567 (12 characters), or (123) 123-4567 (14 characters).

**Example**

The definition of the `OfficePhone` field remains the same as described previously, but each of the three fields (`P1`, `P2`, and `P3`) should be updated to use derivation rules, as this example illustrates for the `P1` field.

```
<defvar name='lenOfficePhone'>
   <length><ref>Office Phone</ref></length>
</defvar>
<Field name='P1'>
   <Display class='Text'>
      <Property name='title' value='Office Phone Number'/>
      <Property name='size' value='3'/>
      <Property name='maxLength' value='3'>
   </Display>
   <Derivation>
      <or>
      <cond><eq>
      <ref>lenOfficePhone</ref>
         <s>8</s></eq>
         <s> </s></eq>
      </cond>
```

```
      <cond><eq>
         <ref>lenOfficePhone</ref>
            <s>12</s></eq>
               <substr>
               <ref>Office Phone</ref>
               <i>0</i>
               <i>1</i>
               </substr>
      </cond>

      <cond><eq>
         <ref>lenOfficePhone</ref>
         <s>14</s></eq>
            <substr>
               <ref>Office Phone</ref>
               <i>0</i>
               <i>1</i>
            </substr>
      </cond>
      </or>
   </Derivation>
</Field>
```

When you are calculating fields, you must consider the data's current format and quality in the resource. It is much easier to ensure the correct field values when creating new users. It is much harder to get existing data to conform to the field when reading it off the resource. You can use derivation rules for any field to check the format of the attribute as it is being read in.

## Recalculating Fields

The system performs field calculations many times when a user is working on a form. The field is calculated when it is first displayed, which sets any default values, and the form is calculated when the user clicks **Save**. Two other actions can cause the form to be evaluated: clicking **Recalculate** on the Edit User page and action fields.

### Example

```
<Field>
   <Display class='Button'>
      <Property name='label' value='Recalculate'/>
      <Property name='command' value='Recalculate'/>
   </Display>
</Field>
```

To ensure that the system recalculates the value of a field, set `action` to `true` in the `Display` class element as shown below:

```
<Display class='Select' action='true'>
```

Add this value only to fields that the user selects or clicks on. Do not add it to test or text area fields. When a field has `action=true` set, the form recalculates this form whenever the field is modified in the browser.

**Example**

```
<Field name='Region'>
   <Display class='Select' action='true'>
      <Property name='title' value='Geographic Region'/>
      <Property name='allowedvalues' value='North, South,
       Central, Midwest'./>
       <Property name='nullValue' value='Select a region'/>
   </Display>
</Field>
```

## Guidelines for Structuring a Form

Use the following guidelines when creating the structure of your new form or editing an existing form.

- **List field elements in the order in which you want them displayed on the page**. The order of the field elements in the form determines the order in which the elements are displayed in the browser.
- **Place the referenced field before the field referencing it**. If a field has an expression that references a value in another field, place the referencing field *after* the referenced field.
- **Disabled fields are ignored when logically true**. If any field defines a Disable expression, it is evaluated. If the result of the Disable expression is logically true, the field will be ignored during form evaluation.

## Optimizing Expressions in Form Fields

Some activities performed in forms can call out to resources external to Identity Manager. Accessing these resources can affect Identity Manager performance, especially if the results are long lists of values (for example, compiling a list of groups or email distribution lists). To improve performance during these types of calls, follow the guidelines in the section titled *Using a Java Class to Obtain Field Data*.

## Example Scenario

The following example illustrates a type of expression optimization.

If you want to query a database for information that is not stored in Identity Manager or accessible as a resource account attribute, follow these general steps:

1. Write a Java class that performs the database access.
2. Define a form field that uses a default expression to call the Java class.
3. Reference the hidden variable.

### Using a Java Class to Obtain Field Data

You will need to write a Java class that has methods that can be called to retrieve information. The example in the following section, *Defining a Hidden Form Field*, uses the `getJobGrade` method, which is a custom method. You should locate this custom class in the `idm\WEB-INF\classes\com\waveset\custom` directory structure. (If these directories do not exist on your system, you must create them.)

Follow these guidelines when writing this class:

- If the method performs an expensive operation, such as a database request, you should make the call in the `Default` expression of a hidden form field. This will cause the value to be stored in the view when the form is first loaded. The value can then be referenced many times without incurring database overhead.
- If the method being called has not been declared static, use the new element to instantiate the class first, as shown in the following example.

### Defining a Hidden Form Field

First define a hidden form field that uses a default expression to call the Java class by not including any `Display` class in the field definition:

```
<Field name='jobGrade'>
   <Default>
      <invoke name='getJobGrade'
         class='com.waveset.custom.DatabaseAccessor'>
         <ref>waveset.accountId</ref>
      </invoke>
   </Default>
</Field>
```

Default expressions are evaluated only if the view does not contain a value for the attribute `jobGrade`. Once the default expression has been run, the result is stored in `jobGrade`, and the expression is not run again.

From the Form Element dialog for the element

1. Select Hidden from the Display Class menu.
2. Click **Ok**.

### Referencing the Hidden Attribute

Once you have defined a hidden attribute, you can reference it in other expressions, such as:

```
<Field name='secureKey'>
   <Disable><lt><ref>jobGrade</ref><i>10</i></lt></Disable>
     ...
</Field>
```

You can use XPRESS `defvar` variables to hold the results of a computation, but the results are typically not as efficient as using a hidden form field.

### Note about Optimizing Variables Beyond a Single Iteration

XPRESS variables typically persist for only a single iteration over the form fields. As a result, you can use a variable within an Expansion phase but not on a subsequent Derivation phase. If you need a computed value to remain relevant beyond one field iteration, use a hidden form field instead. Hidden field values are stored in the view and will persist until the editing session is either canceled or saved.

## Calling Resource Methods from Forms

You can invoke methods on a resource from a form by using the `invoke` method.

The `invoke` method is called by specifying the class name and name of the method. Arguments can also be passed to the method within the invoke tags as shown in the following example.

```
<Default>
   <block>
      <defvar name='vmsResName'>
         <index i='0'>
         <ref>accountInfo.accounts[type=vms].name</ref>
         </index>
      <defvar>
         <invoke name='callResourceMethod'
         class='com.waveset.ui.FormUtil'>
            <ref>display.session</ref>
            <ref>vmsResName</ref>
         <null/>
         </invoke>
</Default>
```

From the Form Element dialog for the field

1. Select Javascript from the Display Class menu.
2. Click **OK**.

## Referencing a Form from Another Form

You can reference particular fields in a separate form (rather than a complete form) through the use of the `<FormRef>` element.

Use the `<FormRef>` element to include another form from within an external form. The following example calls the form named MissingFields.

```
<FormRef name='MissingFields'/>
   <FieldRef name='AuthenticationAnswers'/>
   <FieldRef name='AccountInformation'/>
   <Field name='waveset.backgroundSave'>
      <Display class='Hidden'/>
   </Field>
```

### Using the BPE to Reference Another Form

1. Create a new field by right-clicking in the Property view, then selecting **New → Form Reference**.
2. Double-click on the new Form Reference icon in the tree view. The select element's Form Reference dialog opens.
3. After completing the Name and Base Context fields, click **OK**.

## Referencing Fields from Another Form

You can reference particular fields in a separate form (rather than a complete form) through the use of the `<FieldRef>` element.

Use the `<FieldRef>` element to include a specific field from within an external form. Include:

- **The name of the form in which the field resides**. This must be listed in the include section of the form header with the `<ObjectRef>` element. The type property specifies the name of the form (`UserForm`) and its unique configuration ID. The `name` property identifies the name of the field you will later reference.

### Example

```
<Include>
 <ObjectRef type='UserForm'
  id='#ID#04F5F14E01889DFE:2E5C94:F131DD723D:-7FE4'
  name='Password Library'/>
 <ObjectRef type='UserForm'
   id='#ID#04F5F14E01889DFE:2E5C94:F131DD723D:-7FE3'
  name='Account Summary Library'/>
  <ObjectRef type='UserForm'
   id='#ID#UserForm:UserFormLibrary'/>
         <ObjectRef type='UserForm' name='Global Attributes'/>
</Include>
```

- The field name itself inserted in the section of the form that matches the location on the page you would like it to be displayed.

### Example

```
<Field name='global.fullname' hidden='true'>
    <Expansion>
      <cond>
        <and>
          <ref>global.firstname</ref>
          <ref>global.lastname</ref>
        </and>
        <concat>
          <ref>global.firstname</ref>
             <s> </s>
          <ref>global.lastname</ref>
        </concat>
      </cond>
    </Expansion>
</Field>
```

### Example

In the following example, the `<FieldRef>` element identifies the name of the attribute you want to reference.

```
<Field>
    <Disable>
       <isnull>
          <ref>waveset.id</ref>
       </isnull>
    </Disable>
    <FieldRef name='DynamicChangePasswordFields'/>
</Field>
```

### Using the BPE to Add a Reference to a Field in Another Form

1. Create a new field by right-clicking in the Property view, then selecting **New → Field Reference**.
2. Double-click on the new Field Reference icon in the tree view. The select field reference element's dialog opens.
3. In the Form Element dialog, select FormRef from the Display Class menu.
4. After completing the **Name** and **Base Context** fields, click **OK**.

# Editing a Form

You can edit a form to change its display characteristics or add logical processing to select fields or components. This section divides form-related editing tasks into these two categories:

- **Working with display elements**. This section discusses changing the display characteristics of basic page components when editing an Identity Manager form, especially one that is visible to users. These components include buttons, radio buttons, and checkboxes.
- **Working with hidden components**. These components are the HTML elements you add to Identity Manager forms that are used for background processing or for adding logical processing to visible forms. These elements include the `<Disable>` and `<Expansion>` components and the FormUtil methods.

The HTML components described in this task-oriented section are listed in alphabetical order in the chapter titled *HTML Display Components*.

## Working with Display Elements

The display elements you will most commonly modify or add to an Identity Manager form are buttons, fields, and text entry boxes. Other display elements include tables and section headers.

Any display element that does not have a specified `Display` class will be hidden.

### Using the BPE to Change Display Class

In the BPE, you work on all display elements from the Field dialog associated with the selected field element.

To select an element's display class

1. Double-click the element icon in the tree view.
2. Click the Display Class button.
3. Select a display class from the options menu of available display classes that opens.
4. Click the New Property button, which displays the Select Property popup.
5. From the Select Property popup, select a property you want to assign to the element.
6. Click **OK**. The popup closes, and a field associated with the property you selected appears on the Form Element dialog immediately below the Display class button.
7. Complete the newly displayed fields.

## Buttons

To create a typical push button, use the `<Button>` component.

To align multiple buttons in a horizontal row, use the `<ButtonRow>` component. For more information on this component, see the section titled *Aligning Buttons with <ButtonRow> Component*.

### Example

```
<Field>
   <Display class='Button'>
      <Property name='location' value='true'/>
      <Property name='label' value='Cancel'/>
      <Property name='command' value='Cancel'/>
   </Display>
</Field>
```

To position the button in a button row, include the following code in your button definition: `<Property name='location ' value=' button '/>`. If you do not set this `Property` field, the button will appear in the form in the order in which you include it in the form.

From the Form Element dialog in the BPE

1. Select Button from the Display Class menu.
2. Click **New Property**. The available properties popup is displayed.
3. Select value, then click **OK**. A field labeled **value** displays on the Form element dialog.

## Assigning or Changing a Button Label

When defining a button, its label is identified by the `value` setting in the `label` property as indicated below.

```
<Display class='Button'>
   <Property name='label' value='Cancel'/>
```

The browser displays the preceding code as a button labeled `Cancel`.

From the Form Element dialog in the BPE

4. Select the Button object from the left pane.
5. Right-click and select Edit.
6. Change the appropriate field.

## Overwriting Default Button Names

Two buttons typically are displayed at the bottom of Identity Manager forms. By default, the buttons are labeled **Save** and **Cancel**. To change the names of these buttons, modify the form as follows:

1. On the line that defines the form name (in the header), change the name field

```
   <Form name='Anonymous User Menu'>

   to

  <Form name='Anonymous User Menu' noDefaultButtons=true>
```

2. At the bottom of the form, add the following fields for the **Save** and **Cancel** buttons, and change the labels as desired:

```
<Field>
   <Display class='Button'>
      <Property name='label' value='Submit'/>
      <Property name='name' value='submitButton'/>
      <Property name='value' value='true'/>
      <Property name='command' value='Save'/>
   </Display>
</Field>

<Field>
   <Display class='Button'>
      <Property name='label' value='Cancel'/>
      <Property name='command' value='Cancel'/>
      <Property name='location' value='true'/>
   </Display
</Field>
```

From the Form Element dialog in the BPE, select the No Default Button option.

## Command Values and Buttons

**Note** This section is important only if you are building `Button` objects. If you are building components from XML forms, you can assume that the values in the following table are recognized.

All pages in the Identity Manager interfaces have used the post data parameter named `command` as a mechanism to convey which form submission button was pressed. Page processing systems using components are not required to follow the same convention, but there are some components that contain special support for the *command* parameter, in particular the `Button` component.

Some page processing systems, notably the one that processes XML forms, expect the *command* parameter to be used. Further, several command parameter values have been used to indicate particular actions. These values are described in the following table.

| Parameter | Description |
|---|---|
| `Save` | Indicates that the contents of the form should be saved. |
| `Cancel` | Indicates that contents of the form should be thrown away. |
| `Recalculate` | Indicates that the form should be refreshed based on entered data. |

Any value can be used for the *command* parameter, but you must know which command values are recognized by the page processing system. Posting an unrecognized command value usually results in a redisplay of the page.

## Aligning Buttons with <ButtonRow> Element

To align multiple buttons in a row, use the `ButtonRow` element.

### Example

```
<Field name='OrganizeButtons'>
   <Display class='ButtonRow'>
      <Property name='title' value='Choose a Button'/>
   </Display>
<Field name='ChangePassword'>
   <Display class='Button'>
      <Property name='label' value='Change Password'/>
      <Property name='value' value='Recalculate'/>
   </Display>
</Field>
<Field name='ResetPassword'>
   <Display class='Button'>
      <Property name='label' value='Reset Password'/>
      <Property name='value' value='Recalculate'/>
   </Display>
</Field>
```

## Text Fields

You can include both single-line and multi-line text entry boxes in a form. To create a single-line text entry field, use the `<Text>` element. To create a multi-line text entry field, use the `<TextArea>` element.

**Example of <Text>**

```
<Display class='Text'>
   <Property name='title' value='Zip Code'/>
   <Property name='size' value='10'/>
   <Property name='maxLength' value='10'/>
   <Property name='required' value='true'/>
</Display>
```

### Using the BPE to Work with Text Fields

From the Form Element dialog in the BPE

1. Select Text from the Display Class menu.
2. Click **New Property**. The available properties popup is displayed.
3. Select text, then click **OK**.

### Assigning or Changing a Field Label

When defining a text field or area, its label is identified by the `value` property of the `label` property as indicated below.

```
<Display class='Text'>
   <Property name='label' value='Input'/>
```

The browser displays the preceding code as a text entry field labeled `Input`.

## Containers

Some display elements are contained within components called *container components*. Container components offer a way to:

- Collect multiple components to visually organize in a particular way. Simple containers can concatenate the components horizontally or vertically. Other containers allow more flexible positioning of components and can add ornamentation around the components.
- Group components that you want to hide or disable on a form.

Creating a container class typically results in the generation of an HTML `table` tag.

Typical container components are described in the following table.

| Component | Description |
|---|---|
| <SimpleTable> | Arranges components in a grid with an optional row of column titles at the top |
| <ButtonRow> | Arranges button in a horizontal row. This component is essentially a panel that is preconfigured for horizontal layout. |
| <BorderedPanel> | Positions components into five regions: north, south, east, and west |
| <SortingTable> | Displays a blue and beige table with sortable columns. |

You manipulate container components in the BPE in the same way that you work with other display components.

## Creating a Simple Table

The <SimpleTable> component is a frequently used container component in Identity Manager forms. It arranges components in a grid with an optional row of column titles at the top. The only property for this display component is columns, which assigns column titles and defines the width of the table as defined in a list of strings.

### Example

In the following example, a field that uses SimpleTable to organize several subfields:

```
<Field name='SelectionTable'>

  <Display class='SimpleTable'>
     <Property name='columns'>
        <List>
           <String>Account</String>
           <String>Description</String>
        </List>
     </Property>
  </Display>

  <Field name='accounts[LDAP].selected'>
     <Display class='Checkbox'>
        <Property name='label' value='LDAP'/>
     </Display>
  </Field>
```

```
   <Field>
      <Display class='Label'>
         <Property name='text' value='Primary Corporate LDAP Server'/>
      </Display>
   </Field>

   <Field name='accounts[W2K].selected'>
      <Display class='Checkbox'>
         <Property name='label' value='Windows 2000'/>
      </Display>
   </Field>

   <Field>
      <Display class='Label'>
      <Property name='text' value='Primary Windows 2000 Server'/>
      </Display>
   </Field>

</Field>
```

A browser displays the preceding code as follows:

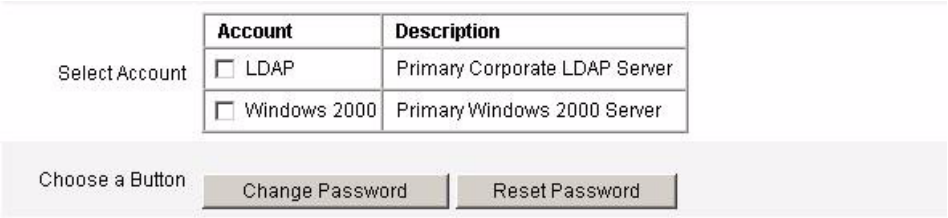



Figure 20. Simple Table

## Grouping Components

To group multiple components on a form to hide or disable them, use the `<SimpleTable>` container as shown in the following example.

### Example

```
<Field>
   <Disable>
      <not>
         <contains>
            <ref>accountInfo.typeNames</ref>
            <s>Windows Active Directory</s>
         </contains>
      </not>
   </Disable>
      <Field name='accounts[AD].HomeDirectory'>
         <Display class='Text'>
            <Property name='title' value='Home Directory'>
         </Display>
      </Field>
</Field>
```

## Working with Lists

The component you use to create a list depends upon list length and whether the user can select more than one option simultaneously.

Text boxes often supply a list of options from which a user can select. These lists are populated by specifying choices within a property called `allowedValues` or by obtaining values dynamically through a method call (FormUtil class methods) to the resource. For information on populating text areas with lists, see the section titled *Populating Lists*.

The following table describes typical list types and the HTML display components used to create them.

| Type of List | HTML Component |
|---|---|
| Option list that offers mutually exclusive values such as true and false | `<CheckBox>`<br>See the section titled *Creating a Checkbox*. |
| Multiple-option list in which users can select only one option | `<RadioButton>`<br>See the section titled *Creating a RadioButton*. |
| Multiple-option list (with many options) in which users can select only one option | `<Select>`<br>See the section titled *Creating a Single-Selection List*. |
| Multiple-option list in which multiple options can be selected simultaneously | `<MultiSelect>`<br>See the section titled *Creating a Multi-Selection List*. |

## Creating a Checkbox

Use the `<Checkbox>` component to display a checkbox. When checked, the box represents a value of `true`. An unselected box represents a `false` value. You can change the checkbox name by editing the value of the `label` property.

### Example 1

```
<Field name='accounts[LDAP].selected'>
   <Display class='Checkbox'>
      <Property name='label' value='LDAP'/>
   </Display>
</Field>
```

### Example 2

```
<Field name='global.Password.Expired'>
   <Display class='CheckBox'>
      <Property name='title' value='User must change password at
      next login'/>
      <Property name='alignment' value='left'/>
   </Display>
</Field>
```

From the Form Element dialog in the BPE

1. Select Checkbox from the Display Class menu.
2. Click **New Property**. The available properties popup is displayed.
3. Select the properties you want to assign, then click **OK**.

## Creating a Radio Button

Use the `<Radio>` component to display a horizontal list of one or more radio buttons. A user can select only one radio button at a time. If the component value is null or does not match any of the allowed values, no button is selected.

### Example

```
<Field name='global.EmployeeType'>
   <Display class='Radio'>
      <Property name='title' value='EmployeeType'/>
      <Property name='labels' value='Employee, Contractor, Temporary,
Part Time'/>
      <Property name='required' value='true'/>
   </Display>
</Field>
```

From the Form Element dialog in the BPE

1. Select Radio from the Display Class menu.
2. Click **New Property**. The available properties popup is displayed.
3. Select radio, then click **OK**.

## Creating a Single-Selection List

Along with the `<MultiSelect>` component, the `<Select>` component provides a list of items to select from. With longer lists of values to select from, the radio buttons can begin to take up precious space on a form. Alternatively, `select` lists can provide a way for the user to select from a long list of possible values. This list supports type-ahead if the list is ordered. You can use the `allowedValues` property to specify the choices from which the user can pick.

### Example

```
<Field name='global.title'>
   <Display class='Select'>
      <Property name='title' value='Title'/>
      <Property name='allowedValues'>
         <List>
         <String>Staff</String>
```

```
         <String>Manager</String>
         <String>Director</String>
         <String>VP</String>
         </List>
      </Property>
   </Display>
</Field>
```

## Creating a Multiselection List

The `<MultiSelect>` component displays a multiselection list box. This textbox displays as a two-part object in which a defined set of values in one box can be moved to a selected box. Values for the list box can be supplied by `allowedValues` elements or obtained dynamically through a method call, such as `getResources`.

Along with the `<Select>` component, the `<MultiSelect>` component can dynamically provide a list of items from which to select. These lists are populated by specifying choices within a property called `allowedValues` or by obtaining values dynamically through a method call to the resource. For information on populating lists within a multiselection entry box, see the section titled *Populating Lists*.

### Example

```
<Field name='waveset.roles'>
   <Display class='MultiSelect' action='true'>
      <Property name='title' value='Roles'/>
      <Property name='availableTitle' value='Available Roles'/>
      <Property name='selectedTitle' value='Current Roles'/>
      <Property name='allowedValues'>
            <invoke name='getObjectNames'
               class='com.waveset.ui.FormUtil'>
               <ref>display.session</ref>
               <s>Role</s>
               <ref>waveset.original.roles</ref>
            </invoke>
      </Property>
   </Display>
</Field>
```

The browser displays a multiselection box as follows:

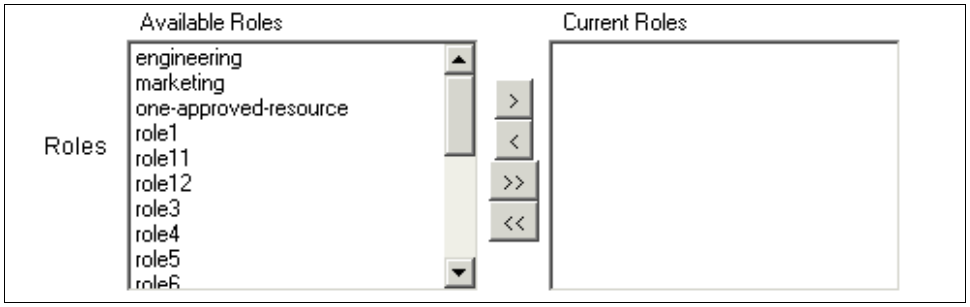



Figure 21. MultiSelection Box in Browser

## Alternative Display Values in a Select List

You can create a `Select` list that displays a different set of values than the values that will actually be assigned to the field. This is often used to provide more recognizable names for cryptic values, or to perform internationalization. This is accomplished by using the `valueMap` property to associate the displayed value with the actual value.

```
<Field name='waveset.organization'>
   <Display class='Select'>
      <Property name='title' value='Add Account'/>
      <Property name='nullLabel' value='Select...'/>
      <Property name='valueMap'>
         <list>
            <s>Top</s>
            <s>Top Level</s>
            <s>Top:OrgB</s>
            <s>Ted's Organization</s>
            <s>Top:OrgC</s>
            <s>Super Secret Org</s>
         </list>
      </Property>
   </Display>
</Field>
```

In the preceding example, the value map is specified as a list of pairs of strings. The odd-numbered strings are the actual values that are assigned to this field. The even-numbered strings are the values that are displayed in the select list. For example, if the select list entry `Ted's Organization` is selected, the value of this field becomes `Top:Orgb`.

## Populating Lists

Lists are frequently populated with options that are dynamically calculated from information that resides in the user object or an external resource. When creating this type of list, you must first create the HTML list components in the form before populating the list. (For additional information on using the HTML text box components, see the sections titled *Creating a Single-Selection List* and *Creating a Multiselection List*.)

There are two ways to populate these lists, including the methods covered in this section:

- Populating lists with the `allowedValues` property
- Using `FormUtil` methods to populate either single-selection or multiselection lists with information dynamically derived from an external resource.

See the section titled *Representing Lists in XML Object Language Instead of XPRESS* for a discussion of the advantages to using XML Object language rather than XPRESS for certain tasks.

### Populating Lists of Allowed Values

The most typical way of populating lists in forms is through the use of the `allowedValues` property. From this property, you can specify an optional list of permitted values for `<Select>` and `<MultiSelect>` elements. The value of this component is always a list and usually contains strings.

#### Example of allowedValues Property

```
<Field name='department'>
   <Display class='Select' action='true'>
      <Property name='title' value='Department'/>
      <Property name='allowedValues'>
         <List>
         <String>Accounting</String>
         <String>Human Resources</String>
         <String>Sales</String>
         <String>Engineering</String>
         </List>
      </Property>
   </Display>
</Field>
```

## Dynamically Populating a Multiselection List of Groups

Multiselection lists typically contain two parts:

- The left side of the list displays the items that are available for selection. These values are defined by the `allowedValues` property. This property can be a list of strings, a list of XML object strings, or a list of strings returned from a call to a Java method.
- The right side of the list displays the items that are currently selected. These values are set by selecting one or more items from the left side's `allowedValues` list and pushing these selections to the selected list. The right side of the list is also populated when the form is loaded and the current settings are retrieved.

### Adding a Multiselection List of Groups

To add a multiselection list of groups that is populated dynamically from the resource

- Add groups to the right side of the schema map. The values displayed in the right side of a text area that displays a multiselection list are populated from the current value of the associated view attribute, which you identify through the field name.
- Add the following text to any form, changing only the `Field name`, `prompt`, `availabletitle`, `selectedtitle`, and the `name` of the resource as needed.

**Note** In the following example, the `:` (colon) that precedes `display.session` indicates that you can ignore the base context of the form and reference objects from the root of the workflow context.

### Example

```
<Field name='global.AD Groups'>
  <Display class='MultiSelect' action='true'>
    <Property name='title' value='AD Group Membership'/>
    <Property name='availableTitle' value='Available AD Groups'/>
    <Property name='selectedTitle' value='Selected AD Groups'/>
    <Property name='allowedValues'>
      <invoke class='com.waveset.ui.FormUtil'
name='listResourceObjects'>
        <!-- send session information which will be used by the method
to validate authorization user -->
        <ref>:display.session</ref>
        <!-- resource object type - This will differ from resource to
resource, but common types are account, group, and "distribution list"
-->
        <s>Group</s>
        <!—- Name of resource being called -->
```

```
        <s>AD Resource Name</s>
        <!-- options map - Some resources have options like the context
that the group is listed in. For example, active directory has multiple
containers. By default, the container used will be the one specified on
the resource. The value can be overridden by specifying it here. If the
resource does not support options, the value should be <null/> -->

       <Map>
         <MapEntry key='context'
value='ou=Austin,ou=Texas,dc=Sun,dc=com'/>
        </Map>
        <!-- cacheList - specify true or false whether you would like
this list to appear in the Resource Object List Cache-->
        <s>true</s>
      </invoke>
    </Property>
  </Display>
</Field>
```

**Notes:**

**If the resource does not support options, the value of `options map` should be `null`. Some resources have options such as the context that the group is listed in. For example, Active Directory has multiple containers. By default, the container used will be the one specified on the resource. This value can be overridden by specifying it here.**

**Specify the value of `cacheList` as `true` or `false` to designate whether this list should be stored in the Resource Object List Cache. This will cause the method to be run once, and the results are stored on the server.**

## Creating a Text Entry Field in a Selection List

There are some conditions under which you'd like to include an option in a selection list in which the user can enter a value instead of choosing from the list. You can create this feature by implementing the three fields as shown in the following example.

- This example creates a selection box with the text string `Other` in it and an adjacent text box. When the user selects the `Other` option from the selection box, the page presents a new field in which the user can enter custom information.
- Implements the `defvar` element to create a variable that defines a list of job positions from which a user can select a relevant position.

**Note** Consider putting into a rule any variables that will be referenced in a form multiple times. In the following example, a list of items to select from is stored in a variable (in the example, `titleList`), which allows the `Derivation` rule to search through it.

The following example is interspersed with descriptive text.

### Example

```
<defvar name='titleList'>
  <list>
    <s>Manager</s>
    <s>Accountant</s>
    <s>Programmer</s>
    <s>Assistant</s>
    <s>Travel Agent</s>
    <s>Other</s>
  </list>
</defvar>
```

The next part of this example contains two visible fields called `title` and `otherTitle`. The `otherTitle` field is displayed only if the user chooses the other option on the selection list. The third hidden field is `global.Title`, which is set from either `Title` or `otherTitle`.

The `Title` field is the main field that the user will select from. If the user cannot find the item that he wants in the list, he can select `Other`. This is a transient field and is not stored or passed to the workflow process when Save is pressed. A Derivation rule is used to send the value from the resource and determine if the value is in the list.

**Note** In the following example, `action` is set to `true` to ensure that form fields populate automatically.

```
<Field name='Title'>
   <Display class='Select' action='true'>
      <Property name='title' value='Title'/>
      <Property name='allowedValues'>
      <Property name='nullLabel' value='Select …"/>
         <expression>
           <ref>titleList</ref>
         </expression>
      </Property>
   </Display>
   <Derivation>
      <cond>
         <isnull><ref>global.Title</ref></isnull>
         <null/>
      <cond>
         <eq>
            <contains>
               <ref>titleList</ref>
               <ref>global.Title</ref>
            </contains>
            <i>1</i>
```

```
         </eq>
         <ref>global.Title</ref>
            <s>Other</s>
      </cond>
      </cond>
   </Derivation>
</Field>
```

The `Other` field will appear on the form only if the user has selected Other from the title field. The value of the `Other` field is set when the form is loaded. It is based upon the value of the `Title` field and the `global.title` field.

```
<Field name='otherTitle'>
    <Display class='Text'>
      <Property name='title' value='Other Title'/>
      <Property name='rowHold' value='true'/>
      <Property name='noWrap' value='true'/>
      <Property name='size' value='15'/>
      <Property name='maxLength' value='25'/>
    </Display>
    <Disable>
      <neq>
        <ref>Title</ref>
        <s>Other</s>
      </neq>
    </Disable>
    <Derivation>
      <cond>
        <eq>
          <ref>Title</ref>
          <s>Other</s>
        </eq>
        <ref>global.Title</ref>
      </cond>
    </Derivation>
</Field>
```

The value of Field is based on the value of the Title field. If the value of this field is set to Other, then the field value is defined by the value of the otherTitle field. Otherwise, it will be the value of the Title field.

```
<Field name='Title'>
   <Expansion>
     <cond>
       <eq>
         <ref>global.fieldTitle</ref>
         <s>Other</s>
       </eq>
       <ref>otherTitle</ref>
       <ref>Title</ref>
     </cond>
   </Expansion>
</Field>
```

## Filtering the List of Resource Accounts before Display in a Form

You can filter the list of resource accounts before displaying them in a form. By default, no filters are applied, except with the Change Password Form in the User Interface, which preserves the default behavior of filtering disabled accounts from the list displayed to the user.

This Exclude filter is defined as a Form property. The filter is a list of one or more AttributeConditions that, when evaluated, determine if a given resource account should be excluded from the displayed list.

### Forms that Support This Feature

The following Forms support the specification of an Exclude filter as a Form property:

Change Password Form (User Interface)

Administrator Interface Forms:

- Change User Password Form
- Deprovision Form
- Disable Form
- Enable Form
- Rename Form
- Reprovision Form
- Reset User Password Form
- Unlock Form

### <Exclude> Property Format

The Exclude Form Property takes the following form:

```
<Configuration wstype='UserForm' ...
   <Extension>
     <Form noDefaultButtons='true'>
  ...
  <Properties>
```

To include disabled resource accounts in the list of displayed accounts, remove the disabled attribute condition from the list.

```
<Property name='Exclude'>
      <list>
        <new class='com.waveset.object.AttributeCondition'>
          <s>disabled</s>
          <s>equals</s>
          <s>yes</s>
        </new>
      </list>
    </Property>
  </Properties>
  ...
</Form>
</Extension>
</Configuration>
```

### Valid View Attributes

The list of valid attribute names are those exposed by the views that are associated with each Form listed above for each instance of a currentResourceAccounts object. Valid attributes include:

- accountDisplayName (string)
- accountId (string)
- directlyAssigned (true/false)
- disabled (yes/no)
- exists (yes/no)
- id (string)
- lastPasswordUpdate (string)
- resource (string)
- selected (true/false)
- type (string)
- userPwdRequired (yes/no)

### Example: Excluding an LDAP Resource Type from a List of Resource Accounts

To exclude from the list of any given form all resource accounts of type LDAP that are not directly assigned, set the Exclude property as follows:

```
<Property name='Exclude'>
  <list>
    <new class='com.waveset.object.AttributeCondition'>
      <s>type</s>
      <s>equals</s>
      <s>LDAP</s>
    </new>
    <new class='com.waveset.object.AttributeCondition'>
      <s>directlyAssigned</s>
      <s>equals</s>
      <s>false</s>
    </new>
  </list>
</Property>
```

## Calling a FormUtil Method from within the allowedValues Property

From within the `allowedValues` property, you can also call `FormUtil` methods that permit you to dynamically retrieve and process information from a resource external to Identity Manager, such as a database.

This example shows how to call a FormUtil method to populate a `<Select>` list. In the following example, the method is called from within the `allowedValues` property. The `getOrganizationsWithPrefixes` method (or any `FormUtil` method) is invoked from within an expression.

### Example

```
<Field name='waveset.organization'>
   <Display class='Select'>
      <Property name='title' value='Organization'/>
      <Property name='autoSelect' value='true'/>
      <Property name='allowedValues'>
            <expression>
            <invoke class='com.waveset.ui.FormUtil'
               name='getOrganizationsWithPrefixes'>
               <ref>:display.session</ref>
            </invoke>
            </expression>
         </Property>
   </Display>
</Field>
```

XPRESS also supports the ability to invoke calls to Java methods from within a resource or ActiveSync adapter. The results of the calls can then be used to populate multiselection or select lists. For information on invoking methods from an expression, see *XPRESS Language*.

## Creating a Label Field

Labels are useful components for displaying the value of a read-only field. Properties of the `<Label>` component permit you to define the display characteristics of the label, including color, value (string), and font style.

**Example**

```
<Field>
   <Display class='Label'>
      <Property name='text' value='Primary Corporate LDAP
         Server'/>
   </Display>
</Field>
```

The `value` attribute is always a string.

# Working with Other Display Elements

Other display elements that you might want to incorporate into a form include:

- section header
- calendar icon
- back link

## Adding a Section Heading to a Form

Section heads are useful to separate sections of long forms with a prominent label. The `<SectionHead>` element displays a new section heading defined by the value of the `title` (`prompt`) property. It is an extension of the `Label` class that sets the `font` property to a style that results in large bold text. It also sets the `pad` property to zero to eliminate the default two-space padding.

**Example**

```
<Field>
   <Display class='SectionHead'>
      <Property name='title' value ='Calculated Fields'/>
   </Display>
</Field>
```

From the Form Element dialog in the BPE

1. Select SectionHead from the Display Class menu.
2. Click **New Property**. The available properties popup is displayed.
3. Select value, then click **OK**. A field labeled **value** displays on the Form element dialog.

## Adding a Calendar Icon to a Form

You can add a calendar icon to a page with the DatePicker element. The user can click this icon to select a calendar date and populate a page field. For example, the Identity Manager Create Audit Report page uses this component to select start and end dates.

The DatePicker element returns a date object. Most resource attributes that you set using DatePicker require a date in the form of a string. The extra text field performs the conversion of the new date object into a string or displays the current setting.

You can obtain the date in one of several formats by passing a different format string to the invoke `dateToString` method as indicated in the following table.

| Expiration Date Field | Format |
|---|---|
| AIX | MMddHHmmyy |
| HPUX | MM/dd/yy |
| Solaris | MM/dd/yyyy |

### Example

```
<Field name='aix_account_expire'>
   <Display class='DatePicker'>
     <Property name='title' value='Set Password Expiration Date'/>
  </Display>
</Field>
```

The field defined below displays the password expiration date as found in the `/etc/security/user` file. It also displays any new date selected by the `aix_account_expire` field if the refresh or recalculate is performed after selecting a new date.

The following code example uses two fields. One field presents the Datepicker calendar to the user, and the other converts the time into the specified format for the resource.

The following field example looks to see if the `aix_account_expire` date field has been set (not null) from the DatePicker field.

If this date field has been set, an `invoke` method is called to convert the date object into a string in the specified format: `MMddHHmmyy`.

Otherwise, display the current date as set on the AIX OS: `accounts[AIX].aix_expires.`

```
<Field name='accounts[AIX].aix_expires'>
  <Display class='Text'>
    <Property name='title' value='Current Password
           Expiration Date'/>
    <Property name='noNewRow' value='true'/>
    <Property name='readOnly' value='true'/>
    <Property name='size' value='10'/>
  </Display>
  <Expansion>
    <cond>
      <notnull>
        <ref>aix_account_expire</ref>
      </notnull>
      <invoke name='dateToString'
             class='com.waveset.util.Util'>
      <!-- First argument to dateToString method is a date
           object -->
        <ref>aix_account_expire</ref>
      <!-- Second argument is the format you want the
           converted date/string in -->
          <s>MMddHHmmyy</s>
      </invoke>
      <ref>accounts[AIX].aix_expires</ref>
    </cond>
  </Expansion>
</Field>
```

From the Form Element dialog in the BPE

1. Select DatePicker from the Display Class menu.
2. Click **New Property**. The available properties popup is displayed.
3. Select value, then click **OK**. A field labeled **value** displays on the Form element dialog.

## Adding a Back Link

You can add a component that behaves the same as the browser **Back** button. This component permits you to add a back link anywhere on the form.

### Example

```
<Field name='back'>
   <Display class='BackLink'>
      <Property name='title' value='Back'/>
      <Property name='value' value='previous page'/>
   </Display>
</Field>
```

From the Form Element dialog in the BPE

1. Select BackLink from the Display Class menu.
2. Click **New Property**. The available properties popup is displayed.
3. Select value, then click **OK**. A field labeled **value** displays on the Form element dialog.

## Positioning Components on a Form

The location of a component on a form is determined by the following factors:

- **The Java Service Page (JSP) associated with this form**. The title and subtitle of the form can be set here.
- **Order in which the components are listed in the form**. The browser will display form fields in the order in which they are included in the form.
- **Use of container forms**. For example, to create a vertical row of buttons, use the `<ButtonRow>` container component.

## Using Hidden Components

Many forms are not visible to the user but help process data from an external resource through the resource adapter before passing it into Identity Manager. In visible forms, too, some components can be hidden. These hidden components are used to process this incoming data as well as to transform data in visible forms.

Some hidden processing within forms is carried out by the methods in the FormUtil Java class. These are frequently used when populating lists in forms from information retrieved dynamically from an external resource.

This section discusses the following tasks, which permit you to process data and optionally hide this processing in forms. Typical tasks include:

- Including XPRESS Logic Using Derivation and Expansion elements
- Calling Methods to Populate Lists
  - Building DN strings
  - Getting a list of object types for which the session owner has access
  - Getting a list of organizations
  - Getting a list of unassigned resources
  - Obtaining a list of resource object names
- Disabling components
- Hiding components

## Including XPRESS Logic Using the Derivation and Expansion Elements

Typically, a field will have either a `Derivation` rule or an `Expansion` rule. If a field includes both types of rules, make sure that these fields do not conflict with each other.

You implement the `<Expansion>` and `<Derivation>` components to use XPRESS to calculate values for form fields. These expressions are similar, differing only in the time at which the expression is evaluated. Derivation rules are typically used to set the value of the field when the form is loaded. Expansion rules are used to set the value of the field whenever the page is recalculated or the form is saved.

| Component | Description | Evaluation |
|---|---|---|
| <Derivation> | Unconditionally calculates an arbitrary value to be used as the value of this field. Whenever a Derivation expression is evaluated, the current field value is replaced. | Derivation rules are run when the form is first loaded or data is fetched from one or more resources. |
| <Expansion> | Unconditionally calculates a value for the field | Expansion rules are run whenever the page is recalculated or the form is saved.<br>For all forms except the User view, Expansion rules are run whenever the page is recalculated or the form is saved. For the User view, an <Expansion> tag runs when the user form is first loaded as well. |
| <Validation> | Determines whether a value entered in a form is valid. | Validation rules are evaluated whenever the form is submitted. |

## Examples of <Derivation> Statement

The following two examples illustrate the potential use for the Derivation

- Example 1: Specifying an authoritative source for a global field
- Example 2: Mapping one set of values into another set

### Example 1

```
<Derivation>
   <or>
      <ref>accounts[AD].fullname</ref>
      <ref>accounts[LDAP].fullname</ref>
   </or>
</Derivation>
```

The preceding example uses the first value, if defined. If the first value is not defined, then it uses the second value.

### Example 2

The following example of using the <Derivation> element shows a field definition that uses *conditional logic* to map one set of values into another set.

In this example, the resource account attribute `accounts[Oracle].locCode` is evaluated against the `AUS` case element first. If it is true, then the second value is the value returned and displayed in the `location` field. If no cases are matched, then the value of the default case is returned. When a matching abbreviation is found as the first expression within a case expression, the value of the second expression within the case expression is returned as the result of the switch expression.

```
<Field name='location' prompt='Location'>
   <Display class='Text'/>
      <Derivation>
      <switch>
            <ref>accounts[Oracle].locCode</ref>
            <case>
               <s>AUS</s>
               <s>Austin</s>
            </case>
            <case>
               <s>HOU</s>
               <s>Houston</s>
            </case>
            <case>
               <s>DAL</s>
               <s>Dallas</s>
            </case>
            <case default='true'>
               <s>unknown</s>
            </case>
      </switch>
      </Derivation>
</Field>
```

## Examples of <Expansion> Statement

The following two examples illustrate the potential use for the `Expansion` element.

- Example 1: Implementing a rule to standardize the case of text entered in a field
- Example 2: Hiding expansion logic

### Example 1

`Expansion` rules transform information that has been entered into a field into values that match the format expected by the resource or established by a rule. For example, a free-form text box in which a user enters a name can include an `Expansion` rule that capitalizes the first initial and lowercases the others.

The use of the global attribute in fields sets any of the resources that have this value when the form is saved. When you load this form, Identity Manager loads the values from each resource (unless the field is disabled). The last resource load sets the value in the form. If a user has made a local change, this change may not show up. Consequently, to ensure that the correct value for the attribute is used, you can use a `Derivation` rule to specify one or more of the resources as an authoritative source for the field.

```
<Field name='global.lastname'>
   <Display class='Text'>
       <Property name='title' value='Last Name'/>
       <Property name='size' value='32'/>
       <Property name='maxLength' value='128'/>
       <Property name='noNewRow' value='true'/>
       <Property name='required'>
         <Boolean>false</Boolean>
       </Property>
   </Display>
        <Expansion>
          <block>
            <defvar name='lname'>
              <downcase>
                <ref>global.lastname</ref>
              </downcase>
            </defvar>
            <defvar name='nlength'>
              <sub>
                <length>
                  <ref>global.lastname</ref>
                </length>
                <s>1</s>
              </sub>
            </defvar>
            <concat>
              <substr>
                <upcase>
                  <ref>global.lastname</ref>
                </upcase>
                <s>0</s>
                <s>1</s>
              </substr>
```

```
            <substr>
              <ref>lname</ref>
              <s>1</s>
              <ref>nlength</ref>
            </substr>
          </concat>
        </block>
      </Expansion>
</Field>
```

As the preceding XPRESS logic could be implemented in multiple fields, consider presenting it in a rule.

**Example 2**

In the following example, this field is also hidden by the absence of any `Display` class definition. The lack of `Display` class definition prevents the field from being displayed in the form, but the field is still considered to be an active part of the form and will generate values for resource attributes through its `<Expansion>` expression.

```
<Field name='accounts[Oracle].locCode'>
   <Expansion>
      <switch>
         <ref>location</ref>
            <case>
               <s>Austin</s>
               <s>AUS</s>
            </case>
            <case>
               <s>Houston</s>
               <s>HOU</s>
            </case>
            <case>
               <s>Dallas</s>
               <s>DAL</s>
            </case>
      </switch>
   </Expansion>
</Field>
```

In this example, it performs the reverse of the mapping performed by the `location` field.

## Example of <Validation> Statement

Validation expressions allow you to specify logic to determine whether a value entered in a form is valid.

The validation expression returns null to indicate success, or a string containing a readable error message to indicate failure. The system displays the validation error message in red text at the top of the form.

The following example contains the logic to determine whether the age entered by user in a field is greater than 0. This expression returns null if the age is greater than or equal to zero.

```
<Field name='age'>
   <Validation>
     <cond>
       <lt>
         <ref>age</ref>
         <i>0</i>
       </lt>
       <s>Age may not be less than zero.</s>
     </cond>
   </Validation>
</Field>
```

## Calling Methods to Populate Lists

Lists in single-selection and multiselection text boxes are often populated with choices that are derived from information from external resources. You can populate lists dynamically with this information by calling one of the FormUtil methods supplied by Sun. These common methods can perform the following tasks:

- Obtain a list of resource object names
- Build DN strings given a name and a base context.
- Get a list of object types for which the session owner has access
- Get a list of organizations
- Get a list of unassigned resources

For information on the `<Select>` and `<MultiSelect>` components and the `allowedValues` property, see the section titled *Populating Lists*.

## Constructing Hash Maps

The `listResourceObjects` and `callResourceMethods` methods accept hash maps. Hash maps can be constructed with the `<Map>` element.

In the following example, the <Map> element builds a static map that never changes.

```
<Map>
   <MapEntry name='key1' value='value1'/>
   <MapEntry name='key2' value='value2'/>
</Map>
```

You can also construct maps with an XPRESS expression through the use of the <map> element. You can use the <map> element to dynamically build a map whose contents are defined by other expressions.

For information on using the XPRESS language to construct a map, see the section titled *Constructing Maps* in the *XPRESS Language* chapter.

## Resource Object Names

To search for or request information on a resource and import it into Identity Manager, you must use object definitions supported by Identity Manager.

The following table lists the object types supported by Identity Manager.

| Supported Object Types | Description |
|---|---|
| account | List of user accounts IDs |
| Administrator_Groups | Names of the administrative groups to which a user can belong |
| Applications | List of applications |
| Distribution Lists | List of email distribution aliases |
| Entitlements | List of PKI entitlements |
| group | List of security and distribution list group objects |
| Group | Security groups |
| groupofNames | List of email Distribution aliases for Exchange 5.5 |
| Nodes | List of SP2 nodes |
| PostOffices | List of GroupWise post offices |
| profile | List of top secret profiles |
| PROFILE | List of Oracle profiles from the DBA_PROFILES table |
| ROLE | List of Oracle roles from the DBA_ROLES table |

| Supported Object Types | Description |
|---|---|
| shell | List of available UNIX shells |
| Template | List of NDS Templates |
| USERS | List of Oracle profiles from the DBA_USERS table |
| UnassignedTokens | List of available unassigned tokens |
| User_Properties | List of user property definitions (properties that can be set on the user) |

## Obtaining a List of Resource Object Names

To obtain a list of object names defined for your particular resource, use the `listResourceObjects` method. You can obtain a list with or without map options. Map options are used only on resources that have a directory structure that permit the filtering of returned values to a single container instead of returning the complete list.

To ensure that you get the resource object list from the resource and not from the server's cache, first invoke the `clearResourceObjectListCache()` method or set the *cacheList* argument to `false`. However, using the cache improves performance on large lists. The resource is contacted only once, and the results are stored on the cache. Consequently, Sun recommends using the cache.

In addition, you can specify a set of one or more key/string value pairs that are specific to the resource from which the object list is being requested.

The following table lists the object types that are supported by each resource.

| Resource | Supported Object Types |
|---|---|
| AIX | account, Group |
| ACF2 | account |
| ClearTrust | account, Group, group, Administrator_Groups, Applications, Entitlements, User_Properties |
| Entrust | Group, Role |
| Exchange5.5 | groupofNames |
| GroupWise | account, Distribution Lists, PostOffices |
| HP-UX | account, Group, shell |

| Resource | Supported Object Types |
|---|---|
| LDAP | account, Group |
| Oracle | USERS, ROLE, PROFILE |
| Natural | account, Group |
| NDS | account, Group |
| PeopleSoft | account |
| RACF | account, Group |
| SAP | account, table, profiles, activitygroups |
| SecurID | UnassignedTokens |
| SP2 | Nodes |
| Solaris | account, Group, shell |
| TopSecret | account |
| VMS | account |
| Windows Active Directory | account, Group<br>You can specify any Active Directory valid object class name as an object type. (A list of object class names can be found in the Active Directory schema documentation.) The list returned contains the distinguished names of the objects. By default, the method searches in the container that is specified by the Container resource attribute. However, you can specify a container as an option to the `listResourceObjects` call. Its value should be the distinguished name of a container. Only objects within that container are listed. |
| Windows NT | account, Group |

## Obtaining a List of Resource Objects without Map Options

To obtain a list of resource objects without map options, specify the resource object type and resource name. Note: Some resources support acting on a subset of a list. You can do this by specifying a starting directory.

### Example

In the following example:

- The `<UnassignedTokens>` string identifies the resource object type that you want to get. Other common resource object types are groups, distribution lists, and accounts.

- The `<SecurID>` string identifies the resource from which the object type is retrieved.
- `null` value indicates no map options.
- value of `true` tells the server to cache the results.

```
<invoke name='listResourceObjects'
   class='com.waveset.ui.FormUtil'>
      <ref>:display.session</ref>
      <s>UnassignedTokens</s>
      <s>SecurID</s>
      <null/>
      <s>false</s>
</invoke>
```

### Obtaining a List of Resource Objects with Map Options

To obtain a list of resource objects with map options, specify the resource object type, resource name, and a map option that defines the directory to start the search in. The resource must be directory-based.

For example, you can get a list of all Active Directory groups in the Software Access directory by building a map option that performs the search in the directory path (`ou=Software Access, dc=mydomain, dc=com`).

### Example

In the following example

- The `Group` string identifies the resource object type that you want to get. Strings that identify resource object types are identified in the table titled Available Resource Object Types.
- The `AD` string identifies the resource name from which to retrieve the object type. Map options specify the directory from which to retrieve the list.
- A value of `true` tells the server to cache the results.
- A value of `false` tells the server not to cache the results.

```
<invoke name='listResourceObjects'
      class='com.waveset.ui.FormUtil'>
         <ref>:display.session</ref>
            <s>Group</s>
            <s>AD</s>
         <Map>
            // This allows you to return a list of groups only in
            and below the specified container/organizational
            unit
         <MapEntry key='container'
         value='LDAP://hostX.domainX.com/cn=Users,dc=domainX,dc=com'/>
         </Map>
            <s>false</s>
</invoke>
```

## Building DN Strings

With a given user ID and base context, you can dynamically build a list of distinguished names or a single distinguished name. This method does not return a list and is typically used within an Expansion rule.

### Building a Dynamic List of DN strings

You can dynamically build a list of DN strings if you specify a user ID and base context.

### Example

The following example shows how to use user IDs and base context to build a dynamic list of DN strings.

The following code first defines the base context to append to users.

```
<Field name='baseMemberContextContractor'>
   <Default>
      <s>ou=Contractors,dc=example,dc=com</s>
   </Default>
</Field>

<Field name='baseMemberContextEmployee'>
   <Default>
      <s>ou=Employees,dc=example,dc=com</s>
   </Default>
</Field>
```

The user of this form enters data in the following field. This is a likely place for providing a dynamically generated list of user IDs.

```
<Field name='userIds'>
   <Display class='TextArea'>
      <Property name='title' value='UserIds'/>
   </Display>
</Field>
```

The following hidden field includes logic that calculates values.

```
<Field name='Members'>
   <Expansion>
      <switch>
      // Look at the role assigned to the users
      <ref>waveset.role</ref>
      <case>
      // If user has "Contractor Role" then build DN like this:
      // ex: CN=jsmith,ou=Contractors,dc=example,dc=com
         <s>Contractor Role</s>
         <invoke name='buildDns' class='com.waveset.ui.FormUtil'>
            <ref>userId</ref>
            <ref>baseMemberContextContractor</ref>
         </invoke>
      </case>
      <case>
      // Otherwise, if user has "Employee Role", then build DN like
         this:
         // ex: CN=jdoe,ou=Employees,dc=example,dc=com
         s>Employee Role</s>

         <invoke name='buildDns' class='com.waveset.ui.FormUtil'>
            <ref>userId</ref>
            <ref>baseMemberContextEmployee</ref>
         </invoke>
         </case>
      </switch>
   </Expansion>
</Field>
```

### Building a Single DN String

You can call the `buildDn` method to populate a list or text area with a single DN.

### Example

```
<invoke name='buildDn' class='com.waveset.ui.FormUtil'>
   <s>jdoe</s>
   <s>dc=example,dc=com</s>
</invoke>
```

This example returns `CN=jdoe,dc=example,dc=com`.

## Getting a List of Unassigned Resources

To retrieve a list of all resources to which the user ID could potentially have permission to view but is currently unassigned, call the `getUnassignedResources` method.

The `<ref>` statements identify the view attribute that contains information about the specified user.

**Example**

```
<invoke name='getUnassignedResources' class='com.waveset.ui.FormUtil'>
   <ref>:display.session</ref>
   <ref>waveset.role</ref>
   <ref>waveset.original.resources</ref>
</invoke>
```

## Retrieving a List of Accessible Object Types

To get a list of object types that the session owner currently has access to, use the `getObjectNames` method.

You can request the following object types:

- Account
- Administrator
- Configuration
- EmailTemplate
- Resource
- Role
- System
- TaskInstance
- User
- UserForm

For a complete list of object types, see the List Objects option on the Debug page.

**Example**

```
<invoke name='getObjectNames' class='com.waveset.ui.FormUtil'>
   <ref>:display.session</ref>
   <s>UserForm</s>
</invoke>
```

## Retrieving a List of Object Types Accessible by the Session Owner

To get a list of object names for which the session owner has access, use the `getObjectNames` method.

### Example

```
<invoke name='getObjectNames' class='com.waveset.ui.FormUtil'>
  <ref>:display.session</ref>
</invoke>
```

## Getting a List of Organizations with Prefixes

To get a list of organizations with prefixes (for example, TOP, TOP:IT, TOP:HR), use the `getOrganizationsWithPrefixes` method.

### Example

```
<invoke name='getOrganizationsWithPrefixes'
class='com.waveset.ui.FormUtil'>
   <ref>:display.session</ref>
</invoke>
```

## Getting a List of Organizations without Prefixes

To retrieve a list of organizations without prefixes (for example, TOP, TOP, TOP), use the `getOrganizations` method.

### Example

```
<invoke name='getOrganizations'
class='com.waveset.ui.FormUtil'>
   <ref>:display.session</ref>
</invoke>
```

## Getting a List of Organizations Display Names with Prefixes

To retrieve a list of organization display names with prefixes, use the `getOrganizationsDisplayNamesWithPrefixes` method.

```
<invoke name='getOrganizationsDisplayNamesWithPrefixes'
class='com.waveset.ui.FormUtil'>
   <ref>:display.session</ref>
</invoke>
```

## Retrieving a List of Applications Unassigned to the User

To get a list of applications to which the user is not currently assigned, use the `getUnassignedApplication` method.

### Example

```
<invoke name='getUnassignedApplications'
class='com.waveset.ui.FormUtil'>
   <ref>:display.session</ref>
   <ref>waveset.roles</ref>
   <ref>waveset.original.applications</ref>
</invoke>
```

## Disabling Fields

When you disable a field, the field (and any fields nested within it) is not displayed in the page, and its value expressions are not evaluated. If the view already contains a value for the disabled field, the value will not be modified.

```
<Disable></Disable>
```

**Note** Keep in mind that `global.*` attributes are derived from enabled fields only. If a form dynamically disables a field (instead of hiding it), this field value will not be available through the `global.*` attributes.

### Example

```
<Disable>
   <eq><ref>userExists</ref><s>true</s></eq>
</Disable>
```

**Note** Disable expressions are evaluated more frequently than other types of expression. For this reason, keep any `Disable` expression relatively simple. Do not call a Java class that performs an expensive computation, such as a database lookup.

## Hiding Fields

When you hide a field, the field (and any fields nested within it) is not displayed on the page, but its value is included in the form processing.

To hide a field, specify that a particular field is hidden by not defining a `Display` property for the field. (This is not conditional.)

```
<Field name='field A'/>
```

# Calculating Values

Methods for dynamically calculating values within forms include:

- Generating field values
- Including rules in forms
- Including XPRESS statements in a form

## Generating Field Values

In some forms, you might want to first display a set of abstract derived fields to the user, then generate a different set of concrete resource account attribute values when the form is submitted. This is known as *form expansion*. Expanded fields are often used in conjunction with derived fields. For an example of `<Derivation>` and `<Expansion>` elements, see the section titled *Using the Derivation and Expansion Elements to Include XPRESS Logic in a Form Field*.

## Including Rules in Forms

In forms, you typically call a rule to calculate the `allowedValues` display property or within a `<Disable>` expression to control field visibility. Within forms, rules could be the most efficient mechanism for storage and reuse of:

- a list of corporate departments
- default values
- a list of office buildings

For a comprehensive discussion of rules, see the chapter titled *Rules* in *Identity Manager Deployment Tools*

## Including XPRESS Statements

The XPRESS language is an XML-based expression and scripting language. Statements written in this language, called *expressions*, are used throughout Identity Manager to add data transformation capabilities to forms and to incorporate state transition logic within Identity Manager objects such as Workflow and forms.

XPRESS is a *functional* language that uses syntax based on XML. Every statement in the language is a function call that takes zero or more arguments and returns a value. Built-in functions are provided, and you can also define new functions. XPRESS also supports the invocation of methods on any Java class and the evaluation of Javascript within an expression.

For a comprehensive discussion of XPRESS features, see *XPRESS Language*.

## Why Use XPRESS?

Expressions are used primarily for the following Identity Manager tasks:

- **Customizing the end-user and administrator forms**. Forms use XPRESS to control the visibility of fields and to transform the data to be displayed and saved.
- **Defining flow of control in Workflow**. Workflow uses XPRESS to define *transition conditions,* which determine the order in which steps in the workflow process are performed.
- **Implementing workflow actions**. Workflow actions can be implemented using XPRESS. Action expressions can perform simple calculations, or call out to Java classes or JavaScript to perform a complex operation.

The expressions contained in these elements can be used throughout Identity Manager.

## Example Expression

In the following example, the `<add>` element represents a call to the XPRESS function named `add`.

```
<add> <ref>counter</ref> <i>10</i> </add>
```

This function is passed two arguments:

- *first argument* – value is determined by calling a function named `ref`. The argument to the `ref` function is a literal string that is assumed to be the name of a variable. The value returned by the `ref` function is the current value of the variable *counter*.
- *second argument* -- value is determined by calling a function named `i`. The argument to the `i` function is a literal string that is an integer. The value that the `i` function returns is the integer 10.

The value returned by the `add` function will then be the result of adding the integer 10 to the current value of the variable *counter*. Every function call returns a value for the next operation to use. For example, if the `ref` call returns the value of the counter, then the `<i>` call returns the integer `10`, then the `<add>` call returns the addition of the two calls.

### Example of Expression Embedded within Form

The following example shows the use of XPRESS logic embedded within a Identity Manager form. XPRESS is used to invoke one of the FormUtil Java methods that will produce the relevant role-related choices for display in the browser. Note that the expression is surrounded by the `<expression>` tag.

```
<Field name='waveset.role'>
   <Display class='Select' action='true'>
      <Property name='title' value='Role'/>
      <Property name='nullLabel' value='None'/>
      <Property name='allowedValues'>
         <expression>
            <invoke class='com.waveset.ui.FormUtil' name='getRoles'>
            <ref>:display.session</ref>
                  <ref>waveset.original.role</ref>
            </invoke>
         </expression>
      </Property>
   </Display>
 </Field>
```

# Adding Guidance Help to Your Form

Identity Manager supplies two types of online help:

- Help, which is task-related help and information available from the Identity Manager menu bar
- Guidance (pop-up help), which is field-level help that is available left of the field or area that is marked with a guidance icon  .

## How to Specify Guidance Help for a Component

You can associate guidance help text with any component, although it is currently displayed only by the `EditForm` container. You can specify guidance text in one of three ways:

- Explicitly assign it the component with the `help` property. See the section titled *Using the help Property*.
- Indirectly assign it to the component with the `helpKey` property, which references an entry in a help catalog. See the section titled *Using the helpKey Property*.
- Indirectly associate it with the component by matching the component's `title` property with an entry in a help catalog. See the section titled *Matching the Component's title Property with a Help Entry*.

## How Identity Manager Evaluates Help Resources

When the system renders a component, it will first check to see if the `help` property is set. If it is, it will use that text. Next, it checks to see if the `helpKey` is set and searches the catalog for an entry with that key. If the `helpKey` property is not set, or there is no matching key in the catalog, the system will then try to use the component title as a help key.

## Using the help Property

The simplest way to define guidance text is to set the `help` property. When using XML forms, this means that the help text will be embedded within a potentially large amount of XML, which makes review and editing of the help text more difficult. An alternative is to use a *help catalog* to store the help text, then reference entries in the catalog from the component.

## Using the helpKey Property

Each entry in a help catalog has a unique name or *key*. One way to reference a catalog entry from a component is by setting the `helpKey` property. When set, the system assumes that this is the value of a key in the current help catalog and loads the associated help text when rendering the component.

## Matching the Component's title Property with a Help Entry

You can also automatically associate help catalog entries with components by using key values in the catalog that are the same as component titles. When using XML forms, a component title can be specified explicitly with a `Property` element. Otherwise, it will be taken from the value of the `prompt` attribute of the containing `Field` element.

## How to Use Help Catalogs

A *help catalog* is a file that contains XML text. The system determines which help catalog to use for a component in the following way:

- The page processing system can specify the catalog by assigning its name to the `helpCatalog` property of the `HtmlPage` object before HTML is generated.
- If no `helpCatalog` property value has been assigned, `HtmlPage` next attempts to locate a catalog through a naming convention that is based upon the request URL. `HtmlPage` takes the URL (typically the URL of a .jsp page), strips off the trailing .jsp and the leading base context, prefixes this with the Identity Manager installation directory, and suffixes it with `-help.xml`.

For XML form designers, the form processing system automatically locates a suitable catalog. Occasionally though, you might prefer more explicit control over which catalog file is used. You can use the `help` attribute of the `Form` element to specify the help catalog to be used by that form. For example:

```
<Form name='Task Launch Form' help='task/common-help.xml'>
       ...
</Form>
```

You might find this useful when sharing a single help catalog among forms that can be referenced through several request URLs.

## Help Catalog Syntax

The syntax of a help catalog file is relatively simple. You must use the document element `WebHelp`, which contains one or more `Page` elements. These `Page` elements in turn contain one or more `Item` elements. Each `Item` element must have a `key` attribute, which is the catalog key for the help text. The content of the `Item` element is the help text.

Although the syntax of the help catalog allows you to specify more than one `Page` element, no page processing system currently supports more than one element.

### Example Simple Help Catalog File

An example of a simple help catalog file follows:

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE WebHelp>

<WebHelp>
   <Page name='user/changeAll.jsp'>

         <Item key='Account Information'>
         Miscellaneous information about your accounts.
         </Item>

         <Item key='Passwords'>
         <![CDATA[
            <b>Passwords</b><br>Enter and confirm a new password.
         ]]>
         </Item>

   </Page>
</WebHelp>
```

### Specifying Help Text within the Item Element

You can specify the help text within the `Item` element in two ways:

- **Working with simple text**. If you are working with simple text, enter the text between the `Item` tags as shown in the `Account Information` item of the preceding example. The browser removes any leading or trailing new line characters in the help text.
- **Including embedded HTML markup in the help text to specify fonts or insert line breaks**. When presenting help text this way, it is usually more convenient to wrap the text in what XML calls a *CDATA marked section* (for example, see the `Passwords` item of the preceding example). The text “[“ is not part of the help text. Instead, they are special XML declarations that cause the parser to treat the content as unparsed character data. This means that elements such as `<b>` found within the marked section are not to be interpreted as XML elements, but as literal text.

If you do not use a `CDATA` marked section, you must "escape" every use of the less-than character in the help text by substituting the string `<` as indicated by the following example:

```
&lt;b>Passwords&lt;/b>&lt;br>Enter and confirm a new password.
```

## Overriding Guidance Help

You can use a custom message catalog to override the guidance text that displays in a pop-up window. If you name your custom message catalog defaultCustomCatalog, Identity Manager recognizes and uses it automatically. Alternatively, you can choose a different name, and then specify that name in System Configuration object under the customMessageCatalog name

For example:

```
<Attribute name='customMessageCatalog' value= 'sampleCustomCatalog' />
```

The following sample custom catalog displays "Waveset Lighthouse 4.1 SP2" as the flyover text for the Help tab.

```
<Waveset>
   <Configuration name="sampleCustomCatalog">
      <Extension>
         <CustomCatalog id="defaultCustomCatalog" enabled="true">
            <MessageSet language="en" country="US">
            <Msg id="UI_END_USER_VERSION">Waveset Lighthouse 4.1
               SP2</Msg>
            </MessageSet>
            <MessageSet language="es" country="ES" variant=
               "Traditional">
            <Msg id="UI_END_USER_VERSION">Sun Lighthouse 4.1 SP2</Msg>
            </MessageSet>
            <MessageSet language="fr">
```

```
            <Msg id="UI_END_USER_VERSION">Waveset Lighthouse 4.1
               SP2</Msg>
            </MessageSet>
         </CustomCatalog>
      </Extension>
   </Configuration>
</Waveset>
```

**Note** To override display of the version number in flyover text, you can replace the UI_END_USER_VERSION message with an empty string.

# Other Form-Related Tasks

Miscellaneous form-related tasks include:

- Invoking the FormUtil methods
- Inserting Javascript into a form
- Testing whether a user or object exists

## Invoking the FormUtil Methods

The FormUtil class is a collection of utility methods that you can call from XPRESS expressions with form objects. They can be used to populate lists of allowed values and validate input. The FormUtil methods are typically called to assist the definition of the allowed values in a list or field.

For a list and description of these methods, see the section titled FormUtil Methods. Use the following syntax to invoke the `FormUtil` methods from within a form:

```
<invoke class = 'com.waveset.ui.FormUtil'
   name = 'listResourceObjects'>
</invoke>
```

where the `name` field identifies the name of the method.

For examples on using these methods within forms, see the section titled *Using Hidden Components*.

## Inserting JavaScript into a Form

To insert pre-formatted Javascript into a form, use the `<JavaScript>` component, especially when using the onClick or onChange properties in components and want to call custom Javascript functions.

The component has an extended property named `script` that can contain the Javascript text.

Alternatively, you can include Javascript by setting the source property. This should be a string that contains a URL fragment relative to the base context. It is the Javascript contained in the indicated file to be loaded by the browser.

### Using the BPE to Insert JavaScript into a Form

From the Form Element dialog in the BPE

1. Select Javascript from the Display Class menu.
2. Enter the JavaScript text into the popup window.
3. Click **Ok**.

## Testing if an Object or User Exists

You might want to check whether an object exists before performing an action. For example, you could look to see if a user name exists in Identity Manager before creating a new user, or validate whether a manager name entered in a field is valid.

To test if an object exists, use the `testObject` method. To specify an object type when using this method, use the object types listed in the section titled *Retrieving a List of Accessible Object Types*. In the following example, the user type is identified as `<s>User</s>`. The second string gives the value of the object type (in this example, `joe`).

**Example**

```
<invoke name='testObject' class='com.waveset.ui.FormUtil'>
   <ref>:display.session</ref>
   <s>User:</s>
   <s>jdoe</s>
</invoke>
```

The `testObject` method returns `true` on successful find of an object. Otherwise, this method returns null.

To test if a user exists, use the `testUser` method. The `<s>` element identifies the name of the user object to find.

**Example**

```
<invoke name='testUser' class='com.waveset.ui.FormUtil'>
   <ref>:display.session</ref>
   <s>jdoe</s>
</invoke>
```

This method returns `true` on successful find. Otherwise, this method returns null.

## Wizard and Tabbed Forms

Both wizard and tabbed forms are mechanisms for structuring unwieldy, single-page forms into more easily managed, multiple-paned forms. Both contain separators between logical sections, or pages. These page separators can be tabs located at the top of the form -- like the tabbed user form -- or a wizard form, which guide the user through the pages using the next/back navigation buttons.

See *Tabbed User Form* later in this chapter for the XML version of the default Tabbed User Form.

### What Is a Wizard Form?

Wizard forms can be a convenient alternative to launching multiple forms from a task when:

- Transition logic between pages is simple
- Privileged system calls between pages are required

Wizard forms contain the two rows of buttons described below.

| Row of buttons | Description |
|---|---|
| top row | Next and Back buttons to traverse through the form panes |
| second row | Contains the standard user form buttons listed in the following table. You can control the second row by setting `noDefaultButtons` option to `true` and implementing your own buttons. |

This second row of button can vary as follows:

| Wizard page | Default buttons |
|---|---|
| first page | Next, Cancel |
| intermediate pages | Prev, Next, Cancel |
| last page | Prev, Ok, Cancel |

### Implementing a Wizard Form

The BPE does not support a visual way to define a tabbed or wizard form containers, so you must configure a tabbed or wizard form using XML.

Wizard form syntax closely resembles tabbed user form structure. To create a wizard form,

- Assign the `WizardPanel` display class to the top-level container (rather than `TabbedPanel`).
- Set the `noCancel` property to `true`.
- Define one or more EditForm fields that contain the pages of the wizard.

The following example provides comments for guidance purposes:

```
<Form>
   <Display class="HtmlPage"/> ----- If not set, causes indentation and
color problems
…
   <Field name='MainTabs'> -- Name of the top container that wraps the
tab pages
      <Display class='TabPanel'/> -- Display class for the top
container: either TabPanel or WizardPanel
   <Field name='Identity'> -- Label of the Tab
      <Display class='EditForm'> -- Each "page" must be an Edit Form
         <Property name='helpKey' value='Identity and Password
Fields'/>
      </Display>

   <Field name='waveset.accountId'>
      <Display class='Text'>
         <Property name='title' value='_FM_ACCOUNT_ID'/>
      </Display>
         <Disable> <ref>waveset.id</ref> </Disable>
   </Field>
         …
   </Field>
…
</Field>
```

## Tips and Workarounds

- Validation errors appear on the last page that the user was on rather than the page on which the attribute appears. To work around this, include information in the validation message to assist the user in navigating back to the correct page.
- For complex wizards, give users some visual clue as to where they are in the process. Using labels or section heads at the top of every page that displays text similar to **Page 1**.
- We do not recommend using conditional navigation in wizard forms. If you must implement it, use Disable expressions for each of the immediate children of the `WizardPanel`. For example:

  ```
  <Field name='Page2'>
  ```

```
    <Display class='EditForm'/>
    <Disable><neq><ref>showPage2</ref><s>true</s></neq></Disable>
...
</Field>
```

- Put fields or buttons on previous pages that cause their gating variables to be set. Disabled pages are automatically removed from transition logic.

# Alternatives to the Default Create and Edit User Forms

When an administrator uses the default User form to edit a user, all resources that are owned by a user are fetched at the moment an administrator begins editing a user account. In environments where users have accounts on many resources, this potentially time-intensive operation can result in performance degradation. If you are deploying Identity Manager in this type of environment, consider using *scalable forms* as an alternative to the default Create and Edit User interfaces.

## Overview: Scalable Forms

*Scalable forms* are customized forms that help improve the performance of Identity Manager's Edit and Create User interfaces in environments with many users and resources. This improved performance results from several features, including:

- *incremental resource fetching*
- *selective browsing* of a user's resources
- *multiple resource editing*

Identity Manager provides scalable versions of the default Edit and Create User forms.

### Incremental Resource Fetching

*Incremental resource fetching* describes one method used by the Identity Manager server to directly query a resource for information over a network connection or by other means. Typically, when an administrator edits a user using the default user form, all resources that are owned by a user are fetched at the moment an administrator begins editing a user account. In contrast, the intent behind the design of scalable forms is to limit fetching by fetching only those resources that the administrator wants to view or modify.

### Selective Browsing

Selective browsing, another feature incorporated into scalable forms, permits an administrator to incrementally view resources based on their owning role, on their resource type, or from a list of resources.

### Multiple Resource Editing

*Multiple resource editing* allows an administrator to select subsets of resources for editing resource attributes. An administrator can select subsets based on roles, resource types, or from a list of resources.

## When to Use Scalable Forms

Consider using scalable forms when

- **Administrators are manually editing users who have many resource accounts**. Implementing a scalable form under these circumstances allows administrators to selectively edit specific resource accounts without incurring the overhead of fetching the user's data for all resource accounts. This mechanism is particularly useful when a certain type of resource responds much slower than the other resource types associated with a user.
- **Custom provisioning processes, such as ActiveSync, target only specific resources for updates**

**Note** Do not use scalable forms when form logic includes attributes that reference other resources. In this configuration, these cross-reference attributes will either not be populated with the latest data, or these resources should be fetched together.

In addition, the scalable version of the Create User form provides limited benefit over the standard default version because a new user has no resources to begin with.

# Available Scalable Forms

Identity Manager ships the following two scalable user forms, which are described below:

- Dynamic Tabbed User form, which provides an alternative to the default Tabbed User form
- Resource Table User form, which provides an alternative to the default Tabbed User form Edit User form

## Dynamic Tabbed User Form

Provides an alternative to the default Tabbed User form, which fetches all resources as soon as an administrator begins editing. In contrast, Dynamic Tabbed User form features incremental fetching and editing of multiple resources based on resource type.

**Note** For detailed implementation information, see the comments associated with each user form in `WSHOME/samples/form_name.xml`.

### Importing and Mapping the Form

Three forms are involved in the substitution of Dynamic Tabbed User form for the default Tabbed User form.

| Form | Description |
|---|---|
| Dynamic Tabbed User Forms | Contains the features of the default Tabbed User Form but dynamically creates one tab per resource type. |
| Dynamic User Forms | Contains fields for creating resource type tabs on the user form. |
| Dynamic Forms Rule Library | Contains the rule library for dynamically printing out attributes for resources that have no specified user form. |
| Dynamic Resource Forms | Contains all forms that are currently compatible with the Dynamic Tabbed User form. Users can customize this list. |

Installing Dynamic Tabbed User form involves two steps: importing the form, and changing the form mapping.

### Step 1: Import the Form

1. From the Identity Manager menu bar, select **Configure → Import Exchange File**.
2. Enter the file name (`dynamicformsinit.xml`) or click **Browse** to locate the `dynamicformsinit.xml` file in the ./sample directory.
3. Click **Import**. Identity Manager responds with a message that indicates that the import was successful.

### Step 2: Change Form Mapping

There are two methods of assigning a user form to an end user. Select a method to edit these form mapping depending upon how administrators in your environment will be using these forms. These methods include:

- **Assign Scalable User Form as the default User Form for all administrators**. If this is your choice, see *Assign Scalable User Form as the Default User Form*. Identity Manager administrators can assign one form that all administrators will use.
- **Separately assign the Scalable User Form to a particular administrator(s)**. If this is your choice, see *Assign Scalable User Form per Administrator*.

### Assign Scalable User Form as the Default User Form

1. From the menu bar, select **Configure → Configure Form and Process Mappings**.
2. In the Form Mappings section, locate `userForm` under the Form Type column.
3. Specify Dynamic Tabbed User Form in the box provided under the Form Name Mapped To column.

### Assign Scalable User Form per Administrator

1. From the menu bar, select **Accounts → Edit User**.
2. Select a user in one of these two ways:
   - Click on user name, then click **Edit**

   or
   - Right-click on the user name to display a pop-up menu, then select the **Edit** menu option
3. After the Default Edit User Form appears, click on the Security tab.
4. Find the User Form field and select Dynamic Tabbed User Form.
5. Click **Save** to save the settings.

## Resource Table User Form

The Resource Table User Form contains most of the driving logic of the scalable version of the Edit User form. This form implements incremental fetching and multiple resource editing based on resource type.

For additional implementation information, see the comments in `WSHOME/samples/resourcetableformsinit.xml`.

## Importing and Mapping the Form

Five forms are involved in the substitution of Resource Table User form for the default Tabbed User form.

| Form | Description |
|---|---|
| Resource Table User Form | Contains all globally available fields that are used for navigation, incremental fetching, and form layout. This main form drives all the other resource-related scalable forms. |
| Resource Table User Form Library | Contains primary fields for the Resource Table User form. Includes bread crumb and navigation fields. |
| Resource Table Account Info Form | Contains Fields for account information section of Resource Table form. |
| Resource Table Rule Library | Contains the rule library for retrieving, counting, analyzing a user's resources. This is mostly used by the User Form Library and to build table data on roles and resources. |
| Resource Table Utility Library | Contains the rules used during the selection process on Resource Table Form, for example these rules retrieve resources per role or per type. |

Installing Resource Table User form involves two steps: importing the form, and changing the form mapping.

### Step 1: Import the Form

1. From the Identity Manager menu bar, select **Configure → Import Exchange File**.
2. Enter the file name or click **Browse** to locate `WSHOME/sample/resourcetableforms.xml`. Importing this file also imports:

### Step 2: Change Form Mapping

1. From the menu bar, select **Configure → Configure Form and Process Mappings**.
2. In the Form Mappings section, locate `userForm` under the Form Type column.
3. Specify Resource Table User Form in the box provided under the Form Name Mapped To column.

## Customizing Scalable Forms

After importing and mapping the scalable user form, you must customize it. To enable incremental fetching, you must identify:

- **resources accounts that are initially fetched**. Use the `TargetResources` form property to represent the resource names to be included on the fetch.
- **operations that are updated** when the final save operation occurs.

Both the Dynamic User Forms and the Resource Table User Forms use resource-specific forms for displaying a user's resource-specific attributes. The following user forms are located in the WSHOME/sample/forms directory and have been adapted for use by scalable forms.

- ./ACF2UserForm.xml
- ./ActivCardUserForm.xml
- ./ADUserForm.xml
- ./AIXUserForm.xml
- ./BlackberryUserForm.xml
- ./ClearTrustUserForm.xml
- ./Exchange55UserForm.xml
- ./HP-UXUserForm.xml
- ./NDSUserForm.xml
- ./OS400UserForm.xml
- ./PeopleSoftCompIntfcUserForm.xml
- ./RACFUserForm.xml
- ./SAPPortalUserForm.xml
- ./SolarisUserForm.xml
- ./SunISUserForm.xml
- ./TopSecretUserForm.xml

These forms are automatically imported along with both Dynamic Tabbed User Forms and Resource Table User forms.

If a deployment is using a resource type other than a type listed above, the scalable forms display a default User form that simply lists all attribute name and values specified in the schema mapping. To use an existing customized resource user form

other than those listed above, you must make certain modifications in order to ensure compatibility with the scalable forms. The following procedure describes some of the steps necessary to ensure compatibility.

**Note** Refer to any one of the forms in this list as an example of this modification.

## Customizing a Resource Form for Compatibility with Scalable User Forms

To add your own customized resource form for use with either the Dynamic Tabbed or Resource Table user forms, follow these general steps.

### Step One: Modify Dynamic Resource Forms

Instructions for adding your own resource form are provided in the `dynamicformsinit.xml` file. Search within this file for the Dynamic Resource Form and follow the steps provide with the form.

**Note** The steps described within the form are presented in comments, and are not displayed in the form once it is imported.

### Step Two: Modify Your Resource Form (if not using one from the list above)

You will need to modify your resource form so that it is compatible. Refer to any of the files listed above for examples. Instructions are listed on the top of each resource form.

# Customizing Tabbed User Form: Moving Password Fields to the Attributes Area

To update two resources with different passwords simultaneously, you must generate a separate password field for each assigned resource. For example, you can have an Exchange password field on the Exchange resource Attribute area (on the Accounts page) that still conforms to password policies that can be set separately from other resources.

## Default Password Policy Display

By default, Identity Manager displays password policy information on the **Accounts > Identity** area, as shown below.

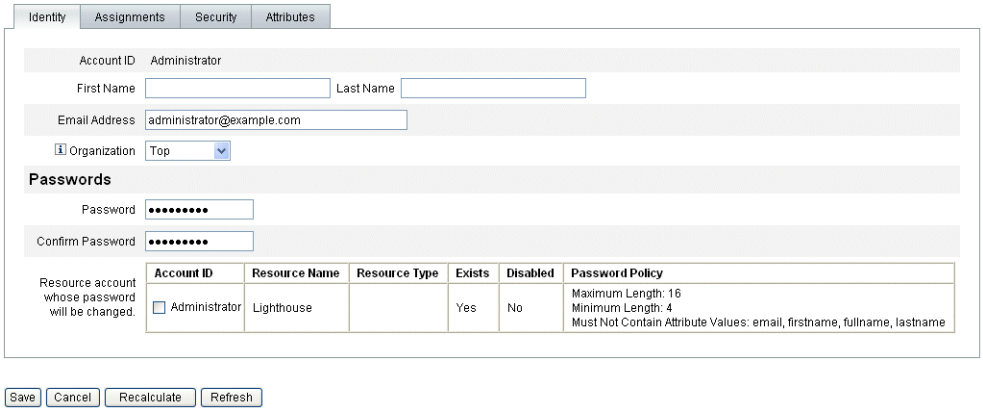



Figure 22. Default Identity Area on the Accounts Page

To move the password fields from their default position on the Identity area to the Attribute area, you must disable the default Identity Manager password synchronization mechanism by following these three steps:

1. Set the `manualPasswordSynchronization` checkout property
2. Add `Field` and `FieldLoop` components to the Tabbed User form
3. Add resource-specific password fields to the Tabbed User form

These steps are described in more detail below.

## Step One: Set the manualPasswordSynchronization Checkout Property

Specify the `manualPasswordSynchronization` view check out option by adding the following property to the form:

```
<Form>
   <Properties>
     <Property name='manualPasswordSynchronization' value='true'/>
...
   </Properties>
...
</Form>
```

When `manualPasswordSynchronization` is set to `true`, Identity Manager displays per-resource password fields rather than using the password synchronizer.

## Step Two: Turn Off Password Synchronization

You can disable password synchronization by turning off the selectAll flag under the Password view. To do this, add the following fields to the default forms:

```
<Field name='password.selectAll'>
   <Comments>
    Force the selectAll flag off so we do not attempt synchronization.
    Necessary because it sometimes is set to true by the view handler.
   </Comments>
     <Expansion><s>false</s></Expansion>
</Field>
   <FieldLoop for='res'>
     <expression>
       <remove>
          <ref>password.targets</ref>
          <s>Lighthouse</s>
       </remove>
     </expression>
   <Comments>
   Also must force the individual selection flags to false and display
   a password prompt for each resource since the view handler will
   default to true for new accounts.
   </Comments>
   <Field name='password.accounts[$(res)].selected'>
   <Expansion><s>false</s></Expansion>
   </Field>
</FieldLoop>
```

## Step Three: Add Resource-Specific Password Fields to Attributes Page

Write resource specific password fields for each resource as follows:

```
<Field name='accounts[resname].password'>
```

# Testing Your Customized Form

You can gather information about edited forms before implementing them in your runtime environment through the following ways:

- Check for errors with the `expression` statements within your form fields through the use of error logging.
- Use the Form Editor to validate the syntax of individual `expression` statements. The Form Editor displays syntax error messages from the parser in a pop-up window. For information on using the Form Editor, see the online help that is associated with the Form Editor.

## Turning On and Off Error Logging

The Identity Manager logging utility reports to standard output any problems with the syntax of form expressions. Once XPRESS tracing is turned on, you can limit log messages to XPRESS statements for a subset of the form with the <block> tag. To obtain more information about the processing of XPRESS statements, a configuration option in the `waveset.properties` file, `xpress.trace,` can be set to true. When this option is set to true, all evaluations of XPRESS statements will generate trace messages to the console. This can be used to debug statements that are evaluated inside a running application whose code cannot be changed to enable tracing through the XPRESS API.

You can turn on XPRESS tracing for all XPRESS fields through either the command line or the Identity Manager Administrator Interface. Turning on tracing this way affects all fields. To limit log messages to a subset of the form, use the `<block>` tag set to limit error tracing to only code within the `<block></block>` tags.

To turn on error logging from the command line for all expression evaluations in Identity Manager:

1. Open the `config/waveset.properties` file for editing.
2. Search on the line `xpress.trace=false`.
3. Change the `false` value to `true`.
4. Save the file.
5. Restart the application server.

Alternatively, you can use the Identity Manager Administrator Interface to turn on and off error logging.

1. Login into Identity Manager as Configurator.
2. Select **Debug** to open the Debug page.
3. From the Debug page, select **Reload Properties**.

To turn tracing off for XPRESS, set the `xpress.trace` value to `false`, and reload the `waveset.properties` file.

# Sample Forms and Form Fields

This section provides examples of the default forms that ship with the product. It also describes how to incorporate sample forms in your environment.

**Note** Note that the versions of forms that ship with your version of Identity Manager may differ slightly from these samples.

- Tabbed User Form
- End User Menu Form
- Anonymous User Menu Form

## User Form Library

A form can be used as a container for a collection of fields rather than being used in its entirety. Identity Manager supports this use of forms with an object called User Form Library, which contains complex fields related to granular resource selection, such as those used for changing passwords.

The following table summarizes each of the libraries associated with User Library

| | |
|---|---|
| **User Library** | The primary user form library. It includes the other libraries in this table and also defines the **AuthenticationAnswers** field for the display and editing of authentication question answers. |
| **Password Library** | Fields related to password specification and synchronization. |
| **Account Summary Library** | Fields that display read-only summary information about the accounts associated with a user. |
| **Account Link Library** | Fields related to account linking, and multiple accounts per resource. |
| **User Security Library** | Fields related to user security including capabilities, form assignment, and approval forwarding. |

## User Form Library

This library contains only fields that are related to the Resource Accounts views which include:

- ChangeUserPassword
- Deprovision
- Disable
- Enable
- Password
- RenameUser
- ResetPassword
- ResetUserPassword
- ResourceAccounts

The library primarily consists of tables that display information about the resource accounts associated with an Identity Manager user and allows them to be selected for various operations.

## Sample Forms

The following table lists the sample forms that are shipped with Identity Manager and their location on the product CD. All are located in the `sample\forms`.

| User Form Name | File Name |
|---|---|
| Active Directory, Active Sync | `ActiveDirectoryActiveSyncForm.xml` |
| AD User Form | `ADUserForm.xml` |
| Domino User Form | `DominoUserForm.xml` |
| GroupWise User Form | `GroupWiseUserForm.xml` |
| HP-UX User Form | `HP-UXUserForm.xml` |
| LDAP Active Sync | `SkeletonDatabaseActiveSyncForm.xml` |
| Linux User Form | `LinuxUserForm.xml` |
| Netegrity Siteminder Admin | `SiteMinderAdminUserForm.xml` |
| Netegrity Siteminder LDAP | `SiteMinderLDAPUserForm.xml` |
| Netegrity Siteminder ExampleTable | `SiteMinderExampleTableUserForm.xml` |
| NDS User Form | `NDSUserForm.xml` |
| NT User Form | `NTform.xml` |
| Open Networks | `ONTUserForm.xml` |
| Oracle ERP | `OracleERPUserForm.xml` |
| OS400 User Form | `OS400UserForm.xml` |
| RACF User Form | `RACFUserForm.xml` |
| SAP | `SAPUserForm.xml` |
| RSA ClearTrust | `ClearTrustUserForm.xml` |
| SecurID User Form | `SecurIDUserForm.xml` |
| Solaris User Form | `SolarisUserForm.xml` |

| User Form Name | File Name |
|---|---|
| AIX User Form | AIXUserForm.xml |
| Exchange55 User Form | Exchange55UserForm.xml |
| Global Attributes | vitalStatform.xml |
| Tivoli Access Manager | AccessManagerUserForm.xml |
| TopSecret | TopSecretUserForm.xml |
| ACF2 | ACF2UserForm.xml |

## Using the Sample Forms Library

You can include the sample forms shipped with Identity Manager in any of the forms you are customizing through the use of the <FormRef> element.

Follow these general steps to add sample forms to your environment:

Step 1: Import the Rule

Step 2: Import the Form

Step 3: Create a New Form from the Default Form (Add Include References and Add the Form Reference)

### Step 1: Import the Rule

Use the Identity Manager Administrator Interface to load the sample rules. To do this:

1. From the Identity Manager menu bar, select **Configure → Import Exchange File**.
2. Enter the sample file name or click **Browse** to locate the file in the idm\sample\rules directory.

   Sample common rule file names are:

   - sample\rules\ListGroups.xml
   - sample\rules\NamingRules.xml
   - sample\rules\RegionalConstants.xml

   Sample resource rule file names are:

   - sample\rules\ADRules.xml
   - sample\rules\NDSRules.xml

- sample\rules\NTRules.xml
- sample\rules\OS400UserFormRules.xml
- sample\rules\RACFUserFormRules.xml
- sample\rules\TopSecretUserFormRules.xml

3. Click **Import**. Identity Manager responds with a message indicating that the import was successful.

## Step 2: Import the Form

Use the Identity Manager Administrator Interface to load the sample form. To do this:

1. From the Identity Manager menu bar, select **Configure → Import Exchange File**.
2. Enter the sample file name or click **Browse** to locate the file in the `idm\sample\forms` directory. Sample form file names are:
   - sample\forms\ACF2UserForm.xml
   - sample\forms\ AIXUserForm.xml
   - sample\forms\Exchange55UserForm.xml
   - sample\forms\HP-UXUserForm.xml
   - sample\forms\NDSUserForm.xml
   - sample\forms\NTform.xml
   - sample\forms\OS400UserForm.xml
   - sample\forms\SecurIDUserForm.xml
   - sample\forms\SolarisUserForm.xml
   - sample\forms\TopSecretUserForm.xml
   - sample\forms\vitalStatform.xml
3. Click **Import**. Identity Manager responds with a message indicating that the import was successful.

## Step 3: Update the Tabbed User Form (Add Include References)

Add an include reference to the sample form from the Tabbed User Form or a main form you created. To do this:

1. Copy the Tabbed User Form and rename it (for example, <***CompanyName***>`tabbedUserForm`).
1. In your Web browser address line, type this URL, and then press **Enter**.

   http://*ApplicationServerHost*:*Port*/idm/debug

   After you authenticate, Identity Manager displays the System Settings page.
2. Select the UserForm option from the Type list, and then click **List Objects**.

3. Click **Edit** next to the <***CompanyName***>`tabbedUserForm` (or the main form you created).
4. Change the includes area of the form to add each sample form, shown in the following example in bold text:

```
<Include>
  <ObjectRef type='UserForm' id='#ID#UserForm:UserformLibrary'
name='UserForm Library'/>
  <ObjectRef type='UserForm' name='UserFormName'/>
<Include>
```

Values for *UserFormName* can be:

- ACF2 User Form
- AIX User Form
- Exchange55 User Form
- HP-UX User Form
- LDAP Active Sync User Form
- Netegrity Siteminder Admin Form
- Netegrity Siteminder LDAP User Form
- Netegrity Siteminder ExampleTable User Form
- NDS User Form
- NT User Form
- Open Networks User Form
- OS400 User Form
- Oracle ERP User Form
- RACF User Form
- RSA ClearTrust User Form
- SecurID User Form
- Skeleton Database Active Sync User Form
- Solaris User Form
- Tivoli Access Manager
- Top Secret User Form
- Global Attributes (vitalStatform.xml)

Continue with the next section before saving the form.

### Step 4: Update the Tabbed User Form (Add the Form)

Add a FormRef for each sample form to add it to the main form.

1. Add the following line for each sample form in an appropriate location in the main form:

   ```
   <FormRef name='UserFormName'/>
   ```

2. Remove the following line:

   ```
   <FormRef name='MissingFields'/>
   ```

3. Click **Save** to save form changes.

# 4 FormUtil Methods

This chapter describes the most commonly used FormUtil methods.

The `FormUtil` class provides a collection of utility methods that are intended to be called from XPRESS expressions within form objects. The FormUtil methods are usually used within the `valueMap` property of `Select` and `MultiSelect` fields to constrain the list of possible values. Additional methods are provided to format string values such as dates and directory DNs.

### Related Information

For examples on using these methods within forms, see the sections titled Using Hidden Components and Populating Lists in *Forms.*

## Invoking Methods

Use the following syntax to invoke the FormUtil methods from within a form:

```
<invoke class = 'com.waveset.ui.FormUtil'
   name = 'method_name'>
      <ref>:display.session</ref>
      <s>arg2</s>
</invoke>
```

where the `name` field identifies the name of the method.

Most FormUtil methods require that a LighthouseContext or Session object be passed as the first argument. That is available within forms by referencing the view attribute `display.session`. Since forms are often used with a base context prefix, it is recommended that the `display.session` reference always be preceded with a colon to remove the base context prefix.

# Methods

This section introduces the most commonly used FormUtil methods.

## callResourceMethod Method

```
callResourceMethod(LighthouseContext s, String resourceName, String
methodName, Map args)  throws WavesetException {
```

### Description

Invokes the specified method on the resource by passing it the specified arguments.

### Parameters

| Parameter | Description |
|---|---|
| ***st*** | Identifies a valid Identity Manager context, typically referred to in forms as `<ref>:display.session</ref>` |
| ***resourceName*** | Specifies the name of the resource on which to call the method |
| ***methodName*** | Indicates the name of the resource method to call |
| ***args*** | Identifies the map of *`args`* that the resource method expects |

# buildDn Method

```
buildDn(String name,
String baseContext)
```

## Description

Takes a name and the base context to append to the name. This method returns a string of fully qualified distinguished (DN) names. For example, passing in `group1` and `dc=example,dc=com` returns the string `cn=group1, dc=example, dc=com`.

## Parameters

| Parameter | Description |
|---|---|
| ***name*** | Specifies the names of the directory object (typically the common name, `cn`). |
| ***baseContext*** | Identifies the base context or container. This is the value of the base distinguished name to which the common name is concatenated to form the full distinguished name of the directory object. |

## Return Value

Returns a single value in the form `CN=<name>,<baseContext>`

## Example

```
<invoke name='buildDn' class='com.waveset.ui.FormUtil'>
   <s>North America</s>
   <s>ou=marketing,dc=acme,dc=com</s>
</invoke>
```

This example returns: `<s>CN=North America,ou=marketing,dc=acme,dc=com</s>`

In XPRESS, this same function can be represented by:

```
<concat>
   <s>CN=</s>
   <s>North America</s>
   <s>,</s>
   <s>ou=marketing,dc=acme,dc=com</s>
</concat>
```

# buildDns Method

```
buildDns(List names,
String baseContext)
```

or

```
buildDns(String names,
String delimiter,
String baseContext)
```

## Description

Takes a list of names and the base context to append to each token in the list. Both variants of this method return a list of fully qualified DN names.

For example, passing in `group1, group2` and `dc=example, dc=com` returns the list: `cn=group1, dc=example, dc=com` and `cn=group2, dc=example, dc=com`

## Parameters

The first variant of this method takes the arguments described in the following table:.

| Parameter | Description |
|---|---|
| ***names*** | Specifies a list of object names |
| ***baseContext*** | Specifies base context |

The second variant of this method takes the three arguments described in the following table.

| Parameter | Description |
|---|---|
| ***names*** | Specifies string containing names that are separated using a delimiter such as a comma or semicolon. |
| ***delimiter*** | Specifies delimiter used in this string of names. The delimiter is typically a coma (,) or a period (.). |
| ***baseContext*** | Identifies base context. |

# Return Values

Returns a list of values or strings, where each value is of the form `CN=<name>,<baseContext>`

## Examples

### Example 1: List buildDns

```
<invoke name='buildDns' class='com.waveset.ui.FormUtil'>
   <list>
      <s>North America</s>
      <s>Europe</s>
   </list>
   <s>ou=marketing,dc=acme,dc=com</s>
</invoke>
```

This example returns:

```
<list>
   <s>CN=North America,ou=marketing,dc=acme,dc=com</s>
   <s>CN=Europe,ou=marketing,dc=acme,dc=com</s>
</list>
```

In XPRESS, this same function can be represented by:

```
<dolist name='commonName'>
   <list>
      <s>North America</s>
      <s>Europe</s>
   </list>
   <concat>
      <s>CN=</s>
      <ref>commonName</ref>
      <s>,</s>
   <s>ou=marketing,dc=acme,dc=com</s>
   </concat>
</dolist>
```

### Example 2: List buildDns(String names, String delimeter, String base-Context)

```
<invoke name='buildDns' class='com.waveset.ui.FormUtil'>
   <s>North America,Europe,China</s>
   <s>,</s>
   <s>ou=marketing,dc=acme,dc=com</s>
</invoke>
```

This example returns:

```
<list>
   <s>CN=North America,ou=marketing,dc=acme,dc=com</s>
   <s>CN=Europe,ou=marketing,dc=acme,dc=com</s>
   <s>CN=China,ou=marketing,dc=acme,dc=com</s>
</list>
```

We do not recommend using XPRESS to provide this functionality.

# checkStringQualityPolicy Method

**checkStringQualityPolicy**`(LighthouseContext s, String policy, Object value, Map map, List pwdhistory, String owner)`

## Description

Checks the value of a designated string against string policy.

## Parameters

| Parameter | Description |
|---|---|
| ***object*** | Identifies the string object |
| ***lighthouseContext*** | Specifies the current user's Lighthouse context |
| ***policy*** | (Required) Specifies the name of the policy that the string will be tested against |
| ***value*** | (Required) Identifies the string value to check |
| ***map*** | (Optional) Provides a map of the data that must not be contained in the string |
| ***pwdhistory*** | (Optional) Lists the user's password history |
| ***owner*** | (Required) Identifies the user whose string value is being checked |

## Return Values

`true` – Indicates that the string passed the policy tests.

`false` – Indicates that the string value does not meet the specified policy.

# controlsAtLeastOneOrganization Method

```
controlsAtLeastOneOrganization(LighthouseContext s, List
organizations)
    throws WavesetException {
```

## Description

Determines whether a currently authenticated user controls any of the organizations specified on a list of one or more organization (ObjectGroup) names. The supported list of organizations include those returned by listing all objects of type ObjectGroup.

## Parameters

| Parameter | Description |
|---|---|
| ***s*** | Specifies current user's Lighthouse context (session) |
| ***organizations*** | Specifies a list of one or more organization names. The supported list of organizations include those returned by listing all objects of type ObjectGroup. |

## Return Values

`true` – Indicates that the current authenticated Identity Manager user controls any one of the organizations in the list.

`false` – Indicates that the current authenticated Identity Manager user does not control any organizations in the list.

# getObject Method

```
getObject(LighthouseContext s,
String typeName,
String id)
   throws WavesetException
```

## Description

Retrieves an object from the repository (subject to authorization).

## Parameters

| Parameter | Description |
|---|---|
| ***s*** | Identifies a valid Identity Manager session, typically referred to in forms as `<ref>:display.session</ref>`. |
| ***typeName*** | Specifies object type. Common object types are User, Object Group, Resource. For a complete list of types, see the Types option list on the Debug page. |
| ***id*** | Specifies ID or name of the object you are looking for. For a User object, the `accountId` would be indicated here. |

# getObjectNames Method

```
getObjectNames(LighthouseContext s,
String typeName)
  throws WavesetException
```

or

```
getObjectNames(LighthouseContext  s,
String typeName,
Map options)
  throws WavesetException
```

## Description

Returns a list of the names of objects of a given type to which the session owner (or currently logged-in user) has access. Additional parameters can be specified in the options map to control the list of names returned.

This method is the preferred way for returning a list of names of objects rather than attempting `session.getObjects()`. This method first goes to the ObjectCache, then to the repository, if necessary, for searches.

## Parameters

| Parameter | Description |
|---|---|
| ***s*** | Identifies a valid Identity Manager session, typically referred to in forms as `<ref>:display.session</ref>`. |
| ***typeName*** | Specifies object type. Common object types are User, Object Group, Resource. For a complete list of types, see the Types option list on the Debug page. |
| ***options*** | See below |

| Option | Value |
|---|---|
| ***conditions*** | See *Additional Options*. |
| ***current*** | See *Additional Options*. |
| ***scopingOrg*** | See *Additional Options*. |

## Return Values

This method returns a list of the names of objects of a given type to which the session owner has access.

# getOrganizationsDisplayNames Method

```
getOrganizationsDisplayNames(LighthouseContext s)
        throws WavesetException
```

or

```
getOrganizationsDisplayNames(LighthouseContext s, Map options)
        throws WavesetException
```

## Description

Returns a list of organization handles that the current administrator has access to. Forms that need select and multiselection lists of organizations should use this method.

**Note** This method defaults to the system configuration setting for `useOrganizationDisplayNames` only if the call to `getOrganizationsDisplayNames()` does not specify a `pathPrefix` option.

## Parameters

Options consist of a map of key-value pair arguments.

| Parameter | Description |
|---|---|
| ***s*** | Identifies a valid Identity Manager session, typically referred to in forms as `<ref>:display.session</ref>`. |
| ***options*** | ***pathprefixes***, ***excluded***, ***current***, ***filterVirtual***, ***conditions***, ***scopingOrg***. See table below. |

| Option | Value |
|---|---|
| ***pathPrefixes*** | `true` – the list of returned handles contains organization paths.<br>`false` – the list of returned handles contains display names<br>not true (unsupplied) – the handle type defaults to the system configuration `useOrganizationsDisplaynames` setting.<br>If your installation of Identity Manager has duplicate organization names, use paths. |
| ***excluded*** | Identifies organizations that will not be included in the return *`valueMap`*. |
| ***current*** | If a list of organization names is passed in for this argument, the return *`list`* includes these extra organizations. If you are requesting paths, this should be a list of paths. If you are requesting display names, it should be a list of display names. |
| ***conditions*** | See *Additional Options*. |
| ***scopingOrg*** | See *Additional Options* |
| ***filterVirtual*** | If this argument is true in the options map, the return *`valueMap`* will not contain any virtual organizations. |

## Version-Specific Behavior

This method behaves differently depending upon whether you have installed the v3.1x version or later releases of Identity Manager. The 3.1.x version of this method does not support multiple organizations with the same name. Version 4.x and greater of this product do support multiple organizations with the same name.

### Why use the v3.1.x Version of this Method?

Reset this method to not support multiple organizations if your customized installation is characterized by the following features:

- Duplicate organization names are not and will not be used, and the preference is to display organizations using the short display names
- Customizations use methods that previously returned the short name of the organizations

- You use the `getOrganizations` method to present the list of available organizations, or it used in evaluation expressions
- References to the waveset.organization view attribute (that previously returned the short name of the org) is used in expressions to set additional resource attributes to view attributes

## Resetting this Method to v3.1.x Behavior

If you are running a version of Identity Manager that is higher than v3.1.x but want the behavior described above, you can edit the following system configuration object attribute as follows:

```
<Attribute name='useOrganizationDisplay Names'>
   <Boolean>false</Boolean>
</Attribute>
```

# getResources Methods

```
getResources(LighthouseContext s)
 throws WavesetException
```

or

```
getResources(LighthouseContext s,
List current)
 throws WavesetException
```

or

```
getResources(LighthouseContext s,
String matchType,
String value)
 throws WavesetException
```

or

```
getResources(LighthouseContext s,
String matchType,
String value
List current)
```

or

```
getResources(LighthouseContext s,
Map Options)
```

## Description

Builds a list of the names of resources that match a particular resource attribute value (such as type=LDAP). If a current list is passed in, the lists are merged.

- The first variant of this method takes `session` only and returns all resources that are visible to the administrator.
- The second variant of this method returns all resources and merges in the current list.
- The third variant of this method returns all resources that match a particular attribute value.
- The fourth variant of this method returns all resources that match a particular attribute value and merges in the current list.

## Parameters

| Parameter | Description |
|---|---|
| ***s*** | Identifies a valid Identity Manager session, typically referred to in forms as `<ref>:display.session</ref>`. |
| ***matchtype*** | Identifies the resource attribute that this method searches when identifying resources to include in the resourced name list. Must be a queryable attribute of the resource object type. Queryable attributes include `type`, `supportsScanning`, `supportsContainerObjectTypes`, and `startupType`, which are described below. |
| ***value*** | Specifies the value of the attribute that this method searches on when identifying resources. |
| ***options*** | current, conditions, scopingOrg. See table. |

For a list of supported resource type names, see *Views*.

| Option | Value |
|---|---|
| ***conditions*** | See *Additional Options*. |
| ***current*** | See *Additional Options*. |
| ***scopingOrg*** | See *Additional Options*. |

Valid queryable attribute types for the `matchType` parameter include the following:

| matchType Parameter | Description |
|---|---|
| `type` | Identifies the resource object type. Valid values are found in the XML Prototype for the resource. For example, the object type that designates Active Directory is `Windows Active Directory`. For Groupwise, it is `GroupWise`. |

| matchType Parameter | Description |
|---|---|
| `supportsScanning` | When set to `true`, specifies that the resource supports scanning. |
| `supportsContainerObjectTypes` | When set to `true`, specifies that the resource supports container objects. |
| `startupType` | Indicates the ActiveSync startup type. Valid values include are "Automatic", "Automatic with failover", "Manual", and "Disabled". |

# getResourceObjects Methods

**getResourceObjects**(LighthouseContext session, String objectType, String resourceId, Map options, String cacheList, String cacheTimeout, String cacheIfExists))

or

**getResourceObjects**(LighthouseContext session, String objectType, String resourceId, Map options)

or

**getResourceObjects**(String subjectString, String objectType, String resourceId, Map options)

or

**getResourceObjects**(String subjectString, String objectType, String resourceId, Map options, String cacheList, String cacheTimeout, String clearCacheIfExists)

## Description

Returns a list of objects where each object contains a set of attributes including type, name, and ID (a DN, or fully qualified name) as well as any requested `searchAttrsToGet` value. The returned value is a List of GenericObjects. Each GenericObject can be accessed similar to how a Map is accessed. Invoking a `get` method on each object, which passes in the name of the attribute, returns the attribute value.

## Parameters

| Parameter | Description |
|---|---|
| ***session*** | Identifies a valid Identity Manager session, typically referred to in forms as `<ref>:display.session</ref>`. |
| ***subjectString*** | Presents the serialized XML representation of the user object. This representation is useful in workflow where there is no current session. For example, to retrieve a subject:<br>`<invoke name='getSubject'>`<br>`  <ref>WF_CONTEXT</ref>`<br>`</invoke>` |
| ***objectType*** | Specifies the name of a valid object class for this specified `resId`. If null, this method returns objects of all object types that are defined by the specified `resId` found within the specified search container and scope. |
| ***resourceId*** | Identifies the resource from which to get the objects. |
| ***cacheList*** | If `true`, then will cache the result on the server. (This is essential for queries that take a long time to return, such as listing profiles on a mainframe). |
| ***cacheTimeout*** | If `true`, sets the number of milliseconds before the cache times out. When the cache times out, the objects from the search results are automatically retrieved from the resource the next time that the request is made (for example, 6000 = 1 minute). |
| ***clearCacheIfExists*** | Forces the cache to be cleared and the objects of the search to be re-fetched from the resource the next time they are requested. |
| ***options*** | Contains name/value pairs that are specific to the resources used to constrain the search. See table below. |

| Option | Value |
|---|---|
| ***searchContext*** | Determines the context in which to perform search (`ResourceAdapter.RA_SEARCH_CONTEXT`). If not specified, the method attempts to get a value from `RA_BASE_CONTEXT`. If no value is specified, this method searches from the logical top. |
| ***searchFilter*** | (Optional) In LDAP search filter format as specified in RFC 1558, of one or more object tuples either AND'ed or OR'ed together. If not specified, the method constructs a filter using the specified *objectType* (`ResourceAdapter.SEARCH_FILTER`). |
| ***searchScope*** | Specifies whether the method should search on the current object within the context of the specified `searchContext`, or in all subcontexts within the specified `searchContext` (`ResourceAdapter.RA_SEARCH_SCOPE`). Valid values are `object`, `oneLevel`, or `subTree`. If object is specified, the `searchContext` is expected to be the full DN of a single object, so that only one object is returned. `oneLevel` indicates that one or more objects should be fetched, starting for a particular `baseContext` (`searchContext`). `subTree` executes the `oneLevel` search recursively. `searchContext` indicates that the search should be performed on all subcontexts within the specified `searchContext`. |
| ***searchTimeLimit*** | Indicates the time limit in milliseconds that a search should not exceed (`ResourceAdapter.RA_SEARCH_TIME_LIMIT`). |
| ***searchAttrsToGet*** | Identifies the list of *objectType* specific attribute names to get per object. |
| ***runAsUser*** | Specifies the user name that this request is to be run as. If not specified, defaults to resource proxy administrative user. |
| ***runAsPassword*** | Specifies the password of `runAsUser`. Required to authenticate with resource in order to run the list request as the specified user. |
| ***current*** | See *Additional Options*. |
| ***conditions*** | See *Additional Options*. |
| ***scopingOrg*** | See *Additional Options*. |

The second flavor of this method uses a ***subjectString*** instead of ***Session***.

# getRoles Method

```
getRoles(LighthouseContext s)
```

or

```
getRoles(LighthouseContext s, String current)
```

or

```
getRoles(LighthouseContext s, List current)
```

or

```
getRoles(LighthouseContext s, Map options)
```

## Description

Returns a list of role names that the current administrator has access to. If a current value or current list is supplied, the role name or names on the list are added to the role names returned.

## Parameters

| Parameter | Description |
|---|---|
| ***s*** | Identifies a valid Identity Manager session, typically referred to in forms as `<ref>:display.session</ref>`. |
| ***options*** | ***current***, ***conditions***, and ***scopingOrg***. See table. |

| Option | Value |
|---|---|
| ***current*** | See *Additional Options*. |
| ***conditions*** | See *Additional Options*. |
| ***scopingOrg*** | See *Additional Options*. |

## Return Values

Returns a list of role names that the current administrator has access to. If a current value or current list is supplied, the role name or names on the list are added to the role names returned.

# getUnassignedApplications Method

```
getUnassignedApplications(LighthouseContext s, Map options)
throws WavesetException {
```

## Description

Builds a list of application names suitable for a user's private applications.(A *private application* is an application that is directly assigned to a user.) This is the list of all accessible applications minus the names of the applications that are already assigned to the user through their role.

The resulting list is convenient for use in forms for assigning private applications.

## Parameters

`getUnassignedApplications` takes the following options:

| Parameter | Description |
|---|---|
| ***context*** | Identity Manager context object |
| ***options*** | ***current***, ***currentRoles***, ***conditions***. See table below. |

| Option | Value |
|---|---|
| ***current*** | List of current application names. The returned list always contains each of these applications regardless of the other options. |
| ***currentRoles*** | List of current role names. The returned list will not contain any applications that are assigned through one of these roles unless that application name is on the current list. |
| ***conditions*** | Can be represented as either a Map or List, as shown below. |

`<map>` first map entry – the name of a queryable attribute supported by this type of object

second map entry – the value an object of this type must have for the associated queryable attribute in order to be returned (the operator is assumed to be `equals`).

If more than one attrname/value pair is specified, they will be logically and'ed together.

`<list>` as a list of AttributeCondition objects.

For example

```
<list>
  <newclass='com.waveset.object.AttributeCondition'>
     <s>MemberObjectGroups</s>
     <s>equals</s>
     <ref>waveset.organization</ref>
   </new>
</list>
```

If more than one AttributeCondition is specified, they will be logically and'ed together.

# getUnassignedResources Method

```
static public List getUnassignedResources(LighthouseContext s, Map
options) throws WavesetException
```

## Description

Build a list of resource names suitable for the private resources of a user. (A *private resource* is a resource that is directly assigned to a user.) This is the list of all accessible resources minus the names of the resources that are already assigned to the user through their role.

The resulting list is convenient for use in forms for assigning private resources.

| Parameter | Description |
|---|---|
| ***context*** | Identity Manager context object |
| ***options*** | ***availableToOrgScope***, ***current***, ***currentRoles***, ***currentResourceGroups***, ***conditions***, ***scopingOrg***. See table below. |

| Option | Value |
|---|---|
| ***availableToOrgScope*** | (List) Specifies organization names (or *paths* or *displayNames*, if unique). Resources that are available to organizations on this list, or an organization below the organization's hierarchy, are returned. Resources on the *current* list are also always returned.). |
| ***current*** | See *Additional Options*. |
| ***currentRoles*** | (List) Specifies current role names.The returned list does *not* contain any resource that is assigned by one of these roles *unless* that resource name is on the current list. |
| ***currentResourceGroups*** | (List) Specifies current resource group names. The returned list does *not* contain any resource that is assigned by one of these resource groups *unless* that resource name is on the current list. |
| ***conditions*** | See *Additional Options*. |
| ***scopingOrg*** | See *Additional Options*. |

## Return Values

This method returns a list of resource names suitable for the private resources of a user.

# getUsers Method

```
getUsers(LighthouseContext s)
```

or

```
getusers(LighthouseContext s, Map options)
```

## Description

The first variant of this method returns all users. The second variant by default returns all users, but you can specify a map of options to further filter the list.

## Parameters

| Parameters | Description |
|---|---|
| ***s*** | Identifies a valid Identity Manager context, typically referred to in forms as `<ref>:display.session</ref>`. |
| ***options*** | ***current***, ***conditions***, ***exclude***, ***scopingOrg***. See table below. |

| Option | Value |
|---|---|
| ***current*** | See *Additional Options*. |
| ***conditions*** | See *Additional Options*.In addition to the supported queryable attribute names for user, you can also use the following pseudo-attributes: `userType` and `similarTo`. See the FormUtil javadoc for more information. |
| ***exclude*** | Specifies which types of users will be excluded from the returned list. Value can be either `administrators` or `endUsers`.<br>`administrators` -- the list of users returned will not contain any users that are either directly or indirectly (via AdminRoles) assigned one or more capabilities and control one or more organizations.<br>`endUsers` -- the list returned will include only users directly or indirectly assigned one or more capabilities and controlled organizations. |
| ***scopingOrg*** | See *Additional Options*. |

# listResourceObjects Methods

```
listResourceObjects(LighthouseContext s,
String objectType,
List resourceList,
Map options,
String cacheList)
```

or

```
listResourceObjects(LighthouseContext s,
String objectType,
List resourceList,
Map options,
String cacheList,
String clearCacheIfExists)
```

The two preceding variants are the same except that the second method clears the cache.

```
listResourceObjects(String subjectString,
String resourceObjectType,
List resourceList,
Map options,
String cacheList)
```

or

```
listResourceObjects(String subjectString,
String objectType,
String resourceId,
Map options,
String cacheList)
String clearCacheIfExists)
```

or

```
listResourceObjects(String subjectString,
String objectType,
String resourceID,
Map options,
String cacheList)
```

or

```
listResourceObjects(String subjectString,
String objectType,
String resourceID,
Map options,
```

```
String cacheList)
String cacheTimeout
String clearCacheIfExists)
```

The two preceding variants are the same except that the second method clears the cache.

```
listResourceObjects(LighthouseContext session,
String objectType,
String resourceId,
Map options,
String cacheList,
String clearCacheIfExists)
        throws WavesetException {
```

or

```
listResourceObjects(LighthouseContext session,
String objectType,
String resourceId,
Map options,
String cacheList)
        throws WavesetException {
```

or

**listResourceObjects**(LighthouseContext session,

```
String objectType,
String resourceId,
Map options,
String cacheList)
String cacheTimeout
String clearCacheIfExists
        throws WavesetException
```

or

```
listResourceObjects (String subjectString, String objectType,
List resourceList, Map options, String cacheList,
String clearCacheIfExists)
```

## Description

Retrieves a list of resource objects of a specified type (for example, `group`). This method first attempts to get the list from the server's `resourceObjectListCache`. If found, this list is returned.

If this list is not found, the method invokes the `listResourceObjects` method on each resource before merging, sorting, and removing duplicates on the resulting lists. Finally, it caches this new list in the server's `resourceObjectListCache` for any subsequent requests for the same resource object type from the same resource(s).

This method runs as the currently authenticated administrator (for example, `subject`). Variants take a single resource ID or a `subject` string and an existing session.

This method has multiple variants that differ on whether:

- The method returns a single resource versus a resource list.
- The cache should be cleared.
- The method is sending a session ID (implemented when the user has already been authenticated) or a subject string (`subjectString`). Typically, you will use `Session`.

## Parameters

| Parameter | Description |
|---|---|
| ***s*** | Identifies a valid Identity Manager session, typically referred to in forms as `<ref>:display.session</ref>`. |
| ***resourceObjectType*** | Identifies the type of resource object class for this specified (for example, `group`). If null, this method returns objects of all object types. |
| ***resourceList*** | List of resource to retrieve the objects from. |
| ***options*** | Contains name/value pairs that are specific to the resources used to constrain the search. |
| ***cacheList*** | Specifies the new list that is stored in the server's `resourceObjectListCache` that is created when the method invokes the `listResourceObjects` method on each resource before merging, sorting, and removing duplicates on the resulting lists. |
| ***clearCacheIfExists*** | Indicates that the cache in the server's `resourceObjectListCache` should be cleared after the method has retrieved the list of resource objects. |

## Return Values

This method returns a list of resource object names of the specified resource object type from the list of resources (`resourceList` of IDs or names).

# testObject Method

```
testObject(LighthouseContext  s,
String typeName,
String id)
```

## Description

Tests to see if a specified object exists, even if the subject is not authorized to view the object. When launching processes to create new users, use this method to prevent attempts to create duplicate objects by an administrator who cannot see the entire tree.

## Parameters

| Parameter | Description |
|---|---|
| ***s*** | Identifies a valid Identity Manager session, typically referred to in forms as `<ref>:display.session</ref>`. |
| ***typeName*** | Specifies the type of object that the method tests for. |
| ***id*** | Identifies the object by either name or ID. If an ID is used, the value of `typeName` is ignored. This parameter cannot be null. |

## Return Values

This method returns:

`true` – object exists

`null` – object does not exist

# testUser Method

```
testUser(LighthouseContext  s,
String id)
```

## Description

Tests to see if a specified user exists, even if the subject is not authorized to view the object. When launching processes to create new users, use this method to prevent attempts to create duplicate objects by an administrator who cannot see the entire tree.

## Parameters

| Parameter | Description |
|---|---|
| ***s*** | Identifies a valid Identity Manager session, typically referred to in forms as `<ref>:display.session</ref>`. |
| ***id*** | Identifies the user by either name or ID.This parameter cannot be null. |

## Return Values

This method returns:

`true` – user exists

`null` – user does not exist

# hasCapability Method

```
hasCapability(LighthouseContext s, String capability)
throws WavesetException {
```

## Description

Checks to see if the user has a specified capability (String). This method checks for a capability that is assigned either directly or indirectly through AdminGroups and/or AdminRoles. Requires a *session* value.

## Parameters

| Parameter | Description |
|---|---|
| ***s*** | Identifies a valid Identity Manager context, typically referred to in forms as <ref>:display.session</ref>. |
| ***capabilities*** | Identifies the capability that will be queried for. |

## Return Values

hasCapability returns:

true – Indicates that the currently authenticated Identity Manager user has the specified capability.

false – Indicates that the current user does not have the specified capability.

# hasCapabilities Method

```
hasCapabilities(LighthouseContext s, List capabilities)
throws WavesetException {
```

## Description

Checks to see if the user has a list of specified capabilities (Strings). These capabilities can be assigned either directly or indirectly through AdminGroups and/or AdminRoles. Requires a *session* value.

## Parameters

| Parameter | Description |
|---|---|
| ***s*** | Identifies a valid Identity Manager context, typically referred to in forms as <ref>:display.session</ref>. |
| ***capabilities*** | Identifies the list of capabilities that will be queried. |

## Return Values

hasCapabilities returns:

true – Indicates that the currently authenticated Identity Manager user has all the specified capabilities.

false – Indicates that the user does not have all the specified capabilities.

# Additional Options

The following options are used by a subset of the FormUtil methods:

- ***scopingOrg***
- ***conditions***
- ***current***

## scopingOrg

Used when two or more AdminRoles are assigned to a user. The value should be the name of an organization. This value specifies that the returned names should contain only ones that are available to organizations that are controlled by an AdminRole that controls the scopingOrg organization and is assigned to the logged-in user.

This option is typically used to ensure that when a user is creating or editing another user, the member organization of the user being edited determines which names (for example, Resourcenames) are available for assignment.

### Using the scopingOrg Parameter

Set this attribute under these conditions:

- The specified user is assigned more than one AdminRole
- You want to ensure that when the administrator is creating or editing a user, the member organization of the user being created or edited determines which object names of the requested type are available for assignment.

For example, if an administrator were assigned both the Engineering AdminRole and Marketing AdminRole, and the administrator is editing a user who is a member of the Engineering organization, the Resources available for assigning to that user should be limited to those available to the organization(s) controlled by the Engineering AdminRole.

### Implementing the scopingOrg Attribute

To implement the behavior described above, add the `scopingOrg` attribute to the `waveset.resources` field in the user form.

Reference the value of the current organization as follows:

```
<Field name='waveset.resources'>
   <Display class='MultiSelect'>
      <Property name='title' value='_FM_PRIVATE_RESOURCES'/>
      <Property name='availableTitle'
          value='_FM_AVAILABLE_RESOURCES'/>
```

```
      <Property name='selectedTitle' value='_FM_SELECTED_RESOURCES'/>
      <Property name='allowedValues'>
         <invoke class='com.waveset.ui.FormUtil'
           name='getUnassignedResources'>
             <ref>:display.session</ref>
               <map>
                  <s>currentRoles</s>
                  <ref>waveset.roles</ref>
                  <s>currentResourceGroups</s>
                  <ref>waveset.applications</ref>
                  <s>current</s>
                  <ref>waveset.original.resources</ref>
                  <s>scopingOrg</s>
                  <ref>waveset.organization</ref>
               </map>
         </invoke>
      </Property>
   </Display>
</Field>
```

## current

Specifies a list of names to be merged with those returned. For example, this is typically the list of selected names in a MultiSelect field to ensure that all selected names are in the MultiSelect's list of available names.

## conditions

This value can be specified in three ways:

| Value Format | Description |
|---|---|
| Map | The <MapEntry> key is the name of a queryable attribute by this type of object and the second is the value an object of this type must have for the associated queryable attribute in order to be returned (the operator is assumed to be "equals"). If more than one attrname/value pairs are specified, they will be logically and'ed together.<br>**Example**<br><Map><br><MapEntry key='memberObjectGroups' value='Top'/><br></Map> |
| map | The first entry is the name of a queryable attribute supported by this type of object. The second entry is the value an object of this type must have for the associated queryable attribute to be returned (the operator is assumed to be "equals").<br>If more than one attrname/value pairs is specified, they will be logically and'ed together.<br>**Example**<br><map><br><s>memberObjectGroups</s><br><ref>waveset.organizations</ref><br></map> |
| list | Lists AttributeCondition objects. If more than one AttributeCondition is specified, they will be logically and'ed together.<br>**Example**<br><list><br><newclass= 'com.waveset.object.AttributeCondition'><br><s>MemberObjectGroups</s><br><s>equals</s><br><ref>waveset.organization</ref><br></new><br></list> |

## Using the conditions Attribute

You can specify a list of one or more object type-specific query attribute conditions to filter the list of names returned by certain FormUtil methods. These methods include methods that take an `options` map as an argument.) You can specify these query attribute conditions as a query option whose key is `conditions` and whose value can be specified as either a map or list of AttributeConditions.

## Examples: Using the condition Attribute to Filter Names

The following examples illustrate the use of the `conditions` attribute to apply additional filters to the list of names returned by a FormUtil method that takes an options map as an argument.

### Example 1

```
<Field name='waveset.resources'>
   <Display class='MultiSelect' action='true'>
      ...
     <Property name='allowedValues'>
       <invoke class='com.waveset.ui.FormUtil'
         name='getUnassignedResources'>
           <ref>:display.session</ref>
               <map>
               <s>currentRoles</s>
               <ref>waveset.roles</ref>
               <s>currentResourceGroups</s>
               <ref>waveset.applications</ref>
               <s>current</s>
               <ref>waveset.original.resources</ref>
               <s>conditions</s>
                 <map>
               <s>supportsContainerObjectTypes</s>
               <s>true</s>
               <s>type</s>
               <s>LDAP</s>
                 </map>
               </map>
       </invoke>
     </Property>
   </Display>
</Field>
```

### Example 2

```
<Field name='orgResource'>
   <Display class='Select' action='true'>
       ...
      <Property name='allowedValues'>
```

```
        <invoke class='com.waveset.ui.FormUtil'
           name='getResourcesSupportingContainerObjectTypes'>
           <ref>:display.session</ref>
             <map>
                <s>conditions</s>
                <list>
                   <new class='com.waveset.object.AttributeCondition'>
                      <s>name</s>
                      <s>starts with</s>
                      <s>ldap</s>
                   </new>
                </list>
             </map>
        </invoke>
      </Property>
   </Display>
</Field>
```

### Example 3

```
<Field name='accounts[Lighthouse].capabilities'>
   <Display class='MultiSelect'>
         ...
     <Property name='allowedValues'>
        <invoke class='com.waveset.ui.FormUtil'
           name='getUnassignedCapabilities'>
           <ref>:display.session</ref>
           <ref>waveset.original.capabilities</ref>
             <map>
             <s>conditions</s>
               <list>
                  <new class='com.waveset.object.AttributeCondition'>
                    <s>name</s>
                    <s>starts with</s>
                    <s>bulk</s>
                  </new>
               </list>
             </map>
        </invoke>
     </Property>
   </Display>
</Field>
```

## Supported Queryable Attribute Names

The list of supported queryable attribute names per object type are categorized as follows:

- all object types

- Account
- AdminGroups
- AdminRole
- Configuration
- Event
- LoginApp
- LoginModGroup
- ObjectGroup
- Policy
- ResourceRole
- WorkItem
- User

Other queryable attribute names are defined in the UIConfig.xml (for example, `firstname` and `lastname`).

## All Object Types

| Queryable Attribute | Description |
|---|---|
| `authType` | Specifies the authorization type, if applicable (for example, for Types such as Configuration or Rule) |
| `id` | Specifies the repository ID for this object |
| `name` | Identifies the name for this object |
| `memberObjectGroups` | Identifies the ObjectGroups that this object is available to or is a member of |

## Account

| Queryable Attribute | Description |
|---|---|
| accountId | Specifies the resource accountId |
| accountExists | Identifies whether account exists (true\|false) |
| disabled | Indicates whether account is disabled (true\|false) |
| discoveredSituation | Specifies the initial status that was discovered during reconciliation. Status includes confirmed and deleted. |
| owner | Specifies the repository ID of this user (userid is displayable name) |
| nativeGUID | Specifies the account GUID, if the resource supports this attribute |
| resource | Identifies the resource name |
| resourceId | Specifies the repository ID of this resource (resource is displayable name) |
| situation | Specifies account status after responses are applied to the discovered situation. Status includes confirmed, missing, deleted. |
| situationChanged | Specifies the date and time that the situation last changed. |
| typeString | Specifies the resource type (for example, LDAP) |
| userid | Uniquely identifies the Identity Manager user with which this account is associated |

## AdminGroup

| Queryable Attribute | Description |
|---|---|
| hidden | Identifies whether an AdminGroup is hidden from display |
| memberAdminGroups | Lists the AdminGroups that are directly assigned to this AdminGroup |

## AdminRole

| Queryable Attribute | Description |
|---|---|
| controlledObjectGroups | Lists the ObjectGroups that are controlled by this Admin Role |
| memberAdminGroups | Lists the AdminGroups that are assigned to this Admin Role |
| adminGroupsRule | Specifies the name of the capabilities rule |
| controlledObjectGroupsRule | Specifies the name of the controlled organizations rule |

## Configuration

| Queryable Attribute | Description |
|---|---|
| configType | Specifies the JAVA class name of the class that extends configuration (for example, UserUIConfig, UserForm) |

## Event

| Queryable Attribute | Description |
|---|---|
| eventType | Specifies the type of event to enable grouping events that are similar |

## LoginApp

| Queryable Attribute | Description |
|---|---|
| hidden | Identifies whether a LoginApp is hidden from display |

## LoginModGroup

| Queryable Attribute | Description |
|---|---|
| `hidden` | Identifies whether a LoginModGroup is hidden from display |

## ObjectGroup

| Queryable Attribute | Description |
|---|---|
| `directoryJunction` | Identifies whether the ObjectGroup is a directory junction (true\|false) |
| `displayName` | Specifies the readable name of the object group's user |

## Policy

| Queryable Attribute | Description |
|---|---|
| `class` | Identifies the JAVA class that implements this policy (for example, `StringQualityPolicy`) |
| `typeString` | Specifies the type of policy (for example, `password` or `accountId`) |

## Resource

| Queryable Attribute | Description |
|---|---|
| `startupType` | Indicates `startupType`. Includes disabled and automatic |
| `supportsContainerObjectTypes` | Indicates whether this resource supports container object types (true\|false) |
| `supportedObjectTypes` | Lists supported object types (for example, group, ou, o, and domain) |
| `supportsScanning` | Indicates whether this resource supports scanning<br>(true\|false) |
| `syncSource` | Indicates whether this resource can be an Active Sync resource<br>(true\|false) |
| `type` | Identifies the resource type (for example, LDAP, AIX, or RACF) |

## Role

| Queryable Attribute | Description |
|---|---|
| `role_resources` | Lists the resources assigned to a role |
| `role_approvers` | Lists the approvers assigned to a role |

## WorkItem

| Queryable Attribute | Description |
|---|---|
| `dirty` | |

| Queryable Attribute | Description |
|---|---|
| itemType | Defines the type of workitem (for example, approval or wizard) |
| owner | Identifies the user that owns this workitem |
| taskid | Identifies the repository ID of the taskinstance that created this workitem |

## User

| Queryable Attribute | Description |
|---|---|
| adminRoles | Specifies the list of AdminRoles assigned to this user |
| controlledObjectGroups | Lists (flattened) all ObjectGroups controlled by this user |
| correlationKey | Identifies the key that is used to identity users during bulk loads |
| dis | Specifies an integer that represents the current disabled state<br>0 indicates no accounts<br>1 indicates some are disabled<br>2 indicates all are disabled |
| lhdis | Indicates whether the Identity Manager user is disabled or not (true\|false) |
| memberAdminGroups | Lists (flattened) all AdminGroups that are assigned to this user |
| prov | Specifies an integer that represents the current provisioning state<br>1 indicates that an update is needed<br>2 indicates OK |
| resourceAccountGuids | Lists resource account GUIDs that are assigned to this user (accountGUID@resourceId) |

| Queryable Attribute | Description |
|---|---|
| `resourceAccountIds` | Lists resource account IDs that are assigned to this user (`accountId@resourceId`) |
| `role` | List the roles that are assigned to this user |
| `user_resources` | Lists the resources that are assigned to this user |

## Supported Operators

- `equals` or `is`
- `notEquals` or `is not`
- `greaterThan` or `greater than`
- `greaterThanOrEqualTo` or `not less than`
- `lessThan` or `less than`
- `lessThanOrEqualTo` or `not greater than`
- `startsWith` or `starts with`
- `endsWith` or `ends with`
- `contains` or `contains'`
- `isPresent` or `exists`
- `notPresent'`
- `isOneOf` or `is one of'`
- `containsAll`

# 5 Identity Manager Views

This chapter introduces Sun Java™ System Identity Manager views, which are data structures used in Identity Manager. It provides background for views, including an overview of how to implement views with Identity Manager workflows and forms as well as reference information.

## Topics in this Chapter

This chapter contains the following sections:

- Understanding Identity Manager Views
- Using the Business Process Editor to Display Views
- Understanding the User View
- Common Views
- View Options
- Deferred Attributes
- Extending Views

## Related Topics

*Identity Manager Forms*

*Identity Manager Workflow*

# Understanding Identity Manager Views

An Identity Manager *view* is a collection of attributes that is assembled from one or more objects managed by Identity Manager. Views are transient, dynamic, and not stored in the repository. The data in a view can change if the view is refreshed to reflect a new role or resource assignment.

If you are using Identity Manager, you will encounter views primarily in forms and workflows. An Identity Manager *form* is an object that describes how to display view attributes in a browser for editing. The form can also contain the rules by which hidden attributes are calculated from the displayed attributes. A *workflow process* is a logical, repeatable, series of activities during which documents, information, or tasks are passed from one participant to another for action, according to a set of procedural rules.

When working with views, it helps to first understand:

- general view concepts
- how views are used in Identity Manager
- frequently customized views

# What Is a View?

The most important view is the *user view*, which contains the user attributes that are stored in Identity Manager and attributes that are read from accounts managed by Identity Manager. Some attributes in the user view are visible in the forms that are presented by the Identity Manager User and Administrator Interfaces. Other attributes are hidden or read-only. Hidden attributes are typically used by rules that derive other visible attributes or calculate field values.

For example, when creating a user (represented as a user view), an administrator enters a first and last name in the appropriate form fields on the Create User page. When the administrator saves the form, the system can calculate the user's full name in a hidden field by concatenating the first and last name. This full name can then be saved to one or more resources, including Identity Manager. Once approved (where approval is required), the system converts the user view back into one or more objects in the Identity Manager repository and sends the view to the resources assigned to the user to create or update the user's resource accounts.

## View Attributes

A view is a collection of name/value pairs that are assembled from one or more objects stored in the repository, or read from resources. The value of a view attribute can be atomic such as a string, a collection such as a list, or reference to another object.

Any Boolean attribute can be omitted from a view. If omitted, the attribute is considered logically false.

# What is a View Handler?

*View handlers* are Java classes that contain the logic necessary to create a view and perform actions specified by setting attributes of the view. View handlers also can include information for the convenience of interactive forms. When a view is checked in, the view handler reads the view attributes and converts them into operations on repository objects. The view handler will often launch a workflow to perform more

complex tasks such as approvals or provisioning. Most view handlers that operate on users prevent you from checking in the view if there is already a workflow in progress for that user.

## Views and Forms

Identity Manager forms contain rules for transforming data in views and describe how the view attributes are to be displayed and edited in a browser. The Identity Manager user interface processes the view and form to generate an HTML form. When the user submits the HTML form, Identity Manager merges the submitted values into the view, then asks the view handler to refresh the view. The view can be refreshed several times during an interactive editing session, and different HTML fields can be generated based on logic in the form. When the user is finished interacting, the view is checked in which typically results in the view being passed as input to a workflow process.

## Views and Workflow

Checking in a view often results in a new workflow process being launched to complete the modifications specified in the view. The workflow can perform time-intensive tasks in the background, launch approval processes, query resources, or take whatever action is appropriate. During approvals, the administrator is able to examine the contents of the view and make changes if desired. After approvals, the view attributes are converted into modifications of one or more repository objects. For views related to users, *provisioning* may occur to propagate the changes to selected resource accounts.

## Common Views

The following views are frequently used with both customized forms and workflows.

| | |
|---|---|
| `User` | Used to manipulate Identity Manager users and provision resource accounts. |
| `AccountCorrelation` | Used to search for users correlating to a specified account (or account attributes). |
| `AdminRole` | Used when assigning an Admin role to a user. |
| `Enable` | Used to present and select the list of resource accounts to be disabled. |
| `Deprovision` | Used to present and select a list of resources to be deprovisioned. |

| | |
|---|---|
| `Disable` | Used to present and select the list of resource accounts to be enabled. |
| `ChangeUserAnswers` | Used to change a user's authentication answers. |
| `ChangeUserCapabilities` | Used to change an Identity Manager user's capabilities. |
| `List` | Used to generate a list of work items and processes in the Identity Manager User Interface. |
| `Org` | Used to specify the type of organization created and options for processing it. |
| `Password` | Used to change an Identity Manager user's password, and optionally propagate the password to resource accounts. |
| `Process` | Used to launch tasks such as workflows or reports. |
| `Reconcile` | Used to request or cancel reconciliation operations. |
| `ReconcileStatus` | Used to obtain the status of the last requested reconciliation operation. |
| `RenameUser` | Used to rename the Identity Manager and resource account identities. |
| `Reprovision` | Used to present and select the list of resources to be reprovisioned. |
| `ResetUserPassword` | Used by administrators to reset a password to a randomly generated password and optionally propagate the new password to resource accounts. |
| `Resource` | Used to manipulate resources. |
| `ResourceObject` | A family of views used to manipulate arbitrary objects supported by a resource, for example groups and mailing lists. |
| `Role` | Used to specify the types of Identity Manager roles created. |
| `TaskSchedule` | Used to create and modify TaskSchedule objects. |
| `Unlock` | Used to unlock accounts for those resources that support native account locking. |
| `WorkItem` | Used when writing a workflow approval form. |
| `WorkItemList` | Used to view information about collections of work items in the repository and to perform operations on multiple work items at a time. |

# Using the Business Process Editor to Display Views

The Business Process Editor (BPE) application is a standalone, Swing-based Java application that provides a graphical view of Identity Manager forms, rules, workflows, and email templates. It also permits the display of view attributes for reference while you customize forms and rules.

**Note** Browsing views in the BPE will reveal both published, supported views and views implemented for internal use only.

To display information about views in the BPE:

- Start the application
- Select Open View, and in the Select View dialog, specify the user or object for which you want to select a view.

The following sections provide specific information and procedures for each of these general steps.

## Starting the Business Process Editor

To run the BPE, you must have:

- Identity Manager installed on your local system
- Configurator-level access to Identity Manager

To start the BPE on either Windows NT or UNIX:

1. From a command line, change to the Identity Manager installation directory.

   Set environment variables with these commands:

   ```
   set WSHOME=<Path_to_idm_directory>
   set JAVA_HOME=<path_to_jdk>
   ```

   **Note** If you are starting the BPE on a UNIX system, you must also enter

   ```
   export WSHOME JAVA_HOME
   ```

2. To start the BPE from the `idm\bin` directory:

   ```
   lh config
   ```

   The BPE interface appears.

# Loading View Information

Unlike other objects that you can display and manipulate in the BPE, views are not objects in Identity Manager that you can access without specifying additional information. To display view information, you must specify the user or object whose information you want to display as well as the type of information.

To load view attributes:

1. Select **File → Open View** from the menu bar. The Select View dialog opens.
2. In the Type field, enter the name of the view type you want to display. (For example, enter **`User`** if you want to see the User View attributes associated with a particular user.) The mostly commonly used views are listed below this procedure in *`Commonly Used Views`*.
3. In the Name field, enter the name of the user or object whose view attributes you want to display. Note: Enter the name of a user whose account information you have access to.
4. Click **OK**. The right pane of the BPE displays the attributes for the specified user or object.

## Commonly Used Views

You can enter one of the following view names in the Types field of the Open View dialog:

- User
- Delete
- Password (for end user password)
- ChangeUserPassword
- Enable
- Disable
- Org
- RenameUser
- Reprovision
- ResetUserPassword
- Role
- WorkItem

# Getting Around in the Business Process Editor

Before you display a view, you should know how to enter information and make selections in the BPE.

The BPE interface includes a menu bar and dialogs for selections. The primary display is divided into two panes: the *tree view* and the *diagram view*.

## Tree View

The tree view (in the left interface pane) shows a hierarchical view of a task, form, view, or rule. In the case of a view, it lists each attribute that belongs to the view.

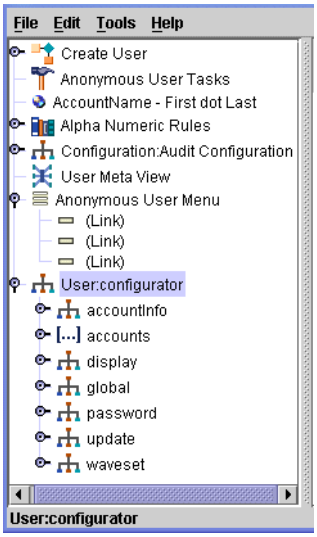



Figure 1. BPE Tree View

### Tree View Icons

Icons that display in the tree view represent the expanded and unexpanded elements of the view.

| This icon… | Represents this process element… |
|---|---|
|  | **view or attribute that contains subattributes (unexpanded)** |
|  | **attribute (with no subattributes)** |
|  | **attribute (expanded)** |

Table 1. View Element Icons

## Main Display of View Attribute Information

The right pane presents the main display of view attributes. View attributes are displayed in a property sheet style as shown below. You can use these for reference as you customize a form or rule.

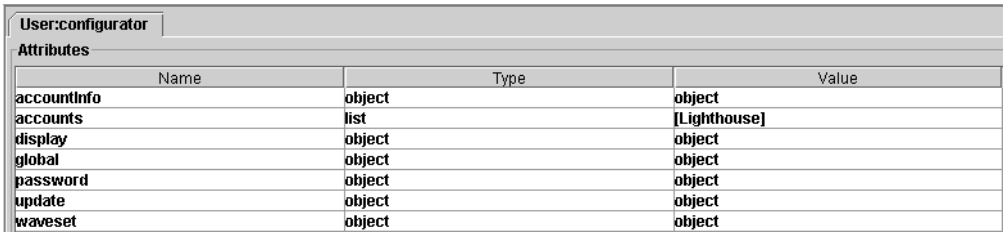



Figure 2. Main Display of View Attribute Information

This display lists all attributes in the view for the selected user or object, including the following basic information about each attribute:

- **Name** — Identifies the name of the view attribute. Double-clicking on an attribute name in the main display opens the attribute popup window, which displays subattributes for this attribute.
- **Type** — Identifies the data type of the attribute. Valid data types include List, Object, and String.
- **Value** — Displays the current value of the attribute.

**Note** The BPE does not currently support editing view attributes.

## Viewing Attribute Information: Attribute Dialog

Each view attribute has an associated dialog that describes the attribute's primary characteristics.

You can display this dialog for any attribute by double-clicking the attribute name in either the tree view or main (diagram) view.

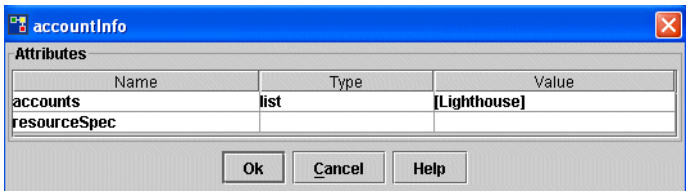



Figure 3. Attribute Popup Dialog

This attribute dialog contains the same type of attribute information that is presented in the main display (right pane). For example, double-clicking on the `accountInfo` attribute name in the display of attributes in the right-hand pane opens the `accountInfo` pop-up window.

# Understanding the User View

The *User view* is the collection of attributes that contain information about an Identity Manager user, including:

- Attributes stored in the Identity Manager repository
- Attributes fetched from resource accounts
- Information derived from other sources such as resources, roles, and organizations

The user view is most often used with forms that are designed for the pages that create or edit users. These pages launch workflow processes that store a changed user view until it is necessary to push the updated view information back out to Identity Manager and associated resources. While the user view is stored in a workflow process, the workflow process can manipulate attribute values through *workflow actions*. Workflow can also expose attribute values for user input through manual actions and approval forms.

## How the User View Is Integrated with Forms

The user view is often used in conjunction with a form. Forms contain rules that control how data is presented through HTML fields and is processed after the HTML page rendering the form is submitted. A system component called the *form generator* combines a form definition and a view to produce HTML that a browser then displays.

View attribute values are displayed by assigning them to an *HTML component* in the form. (See *HTML Display Components* for more information on how view attributes can be displayed.)

Views are implemented as instances of the `GenericObject` class. This class provides a mechanism for the representation of name/value pairs and utilities for traversing complex hierarchies of objects through *path expressions.* A path expression is a string that is interpreted at runtime to traverse an object hierarchy and retrieve or assign the value of an attribute.

You must understand how to write path expressions to assign valid form field names. For more information on using path expressions, refer to the section titled *Path Expressions*.

## How the User View Is Integrated with Workflow

Workflow processes that contain a user view typically store it in a workflow variable named `user`. You can reference a view in the workflow expressions by prefixing `user` to a user view path (for example, `user.waveset.accountId`). The string `waveset` identifies the attribute named `accountId` as belonging to another object named `waveset`, which itself belongs to the user view object.

Approval forms are written for a view known as the *WorkItem view*. The Work Item view by default contains all the workflow variables under an attribute named `variables`. If the approval form is written for a workflow that contains a user view, the prefix `variables.user.` is used to reference attributes in the user view (for example, `variables.user.waveset.roles`). See *WorkItem View* later in this chapter for more information.

## Generic Object Class

At a high level, objects are simply named collections of attributes, which are name/value pairs. The value of an attribute can be an atomic value such as a string, a collection such as a list, or a reference to another object. You can represent almost any object abstractly with the `Map`, `List`, and `String` Java classes.

Within the Identity Manager system, the `GenericObject` class provides a simple memory model for the representation of arbitrary objects and collections. It includes features for easily navigating object hierarchies to access or modify attribute values.

The `GenericObject` class implements the `java.util.Map` interface and internally uses a `java.util.HashMap` to manage a collection of name/value pairs. The entries in this map are called *attributes*. The value of an attribute can be any Java object that is able to serialize itself as XML. The most common attribute values found in a GenericObject:

The following are instances of the following classes:

- `String`
- `Integer`
- `Boolean`
- `EncryptedData`
- `List`
- `Date`
- `GenericObject`
- `X509cert`

You can construct complex hierarchies of objects by assigning `Lists` or `GenericObjects` as attribute values. Once you have assigned attribute values, you traverse this hierarchy to access the values of an attribute.

# Path Expressions

A *path expression* is a string that is interpreted at runtime by the `GenericObject` class to traverse an object hierarchy and retrieve or assign the value of an attribute. Identity Manager uses a system of dots and brackets to represent objects and attributes in the hierarchy.

You use path expressions as the value of the `name` attribute in form fields when customizing a form (for example, `<Field name='user.waveset.roles'/>`).

## Traversing Objects

The following simple example illustrates a `GenericObject` with two attributes:

- `name` (String)
- `address` (GenericObject) The address object, in turn, has an attribute named `street`, which is a string.

To create a path expression to the `street` attribute of the address object, use `address.street`.

Path expressions use the dot character (.) to indicate traversal from one object to another. This is similar to the way dot is used in Java or the '->' operator is used in C. Paths can be long, as illustrated by this example:

```
user.role.approver.department.name
```

## Traversing Lists

You can also use path expressions to traverse values that are lists. Consider an object that has an attribute `children` whose value is a `java.util.List`. Each object in the list is itself a `GenericObject` with a `name` attribute and an `age` attribute. Write the path to the name of the first child as:

```
children[#0].name
```

Path expressions use square brackets to indicate the indexing of a list. The token between brackets is the *index expression*. In the simplest case, this is a positive integer that is used to index the list by element position.

Typically, the position of an object in a list is arbitrary. Index expressions can also specify simple search criteria to identify one object in the list. Objects in a list typically have a `name` attribute, which serves to uniquely identify this object among its peers. Path expressions support an implicit reference to an object's `name` attribute within the index expression.

For example:

```
children[hannah].age
```

The preceding path expression obtains the list of objects stored under the `children` attribute. This list is searched until an object with a `name` attribute equal to `hannah` is found. If a matching object is found, the value of the `age` attribute is returned. The previous example is shorthand for the more general form.

```
children[name=hannah].age
```

### Calculating Lists

You can also write path expressions that calculate `List` values that are not stored in the object. For example:

```
accounts[*].name
```

When an asterisk is found as an index expression, it implies an iteration over each element of the list. The result of the expression is a list that contains the results of applying the remaining path expression to each element of the list. In the previous example, the result would be a list of `String` objects. The strings would be taken from the `name` attribute of each object in the `accounts` list.

Path expressions with * (asterisk) are commonly used with the `FieldLoop` construct in forms to replicate a collection of fields.

## User View Attributes

Whenever you create or modify a user account from a web browser, you are indirectly working with the user view. From the perspective of altering user account information, it is the most significant view in the Identity Manager system.

Workflow processes also interact with the user view. When a request is passed to a workflow process, the attributes are sent to the process as a view. When a manual process is requested during a workflow process, the attributes in the user view can be displayed and modified further.

## MetaView Attributes

If your deployment uses *Identity attributes*, Identity Manager creates an additional namespace in the User view. This additional namespace, called *metaView*, contains identity attribute-related information. Identity Manager creates this MetaView object to store meta view/identity attribute information in the Identity Manager repository.

For each Identity attribute that is defined, there is an attribute in the `metaView` namespace that contains the value of this attribute. For example, for the `firstname`, `lastname`, and `waveset.roles` Identity attributes, the User view has corresponding attributes called `metaView.firstname`, `metaView.lastname`, and `metaView.waveset.roles` that contain the calculated values for each of these attributes.

For more information, see *Working with Attributes* in the *Identity Manager Technical Deployment Overview*

## Introduction

Like all views, the user view is implemented as a `GenericObject` that contains a set of attributes. The values of the attributes in the root object are themselves `GenericObjects`. Attributes can be nested.

The user view contains the attributes described in the following table, which are further defined in subsequent sections.

| Attribute | Description |
|---|---|
| waveset | Contains information stored in the Identity Manager repository (the WSUser object). This is sometimes referred to as the *basic view*. |
| accounts | Contains the values of all resource account attributes fetched from resources. These are typically the values that are edited with forms. |
| accountInfo | Contains read-only information about the resources and accounts associated with the user. |
| display | Contains the read-only runtime state for the interface. It is used only during interactive editing of the user. display.session describes login and access information. display.subject identifies the account under which the user is logged in. display.eventType indicates whether the user view is servicing a create or an update operation. |
| global | Contains attributes that are synchronized across all resource accounts. |
| password | Contains attribute values that are specific to the user's password, password expiration, and target systems. |

Table 2. Top-Level User View Attributes

When you design a form, the field names are typically paths into the user view objects waveset. global, and account attributes (for example, global.firstname).

## Selecting the Appropriate Variable Namespaces

The user view provides several namespaces for deriving account-related information. The following table summarizes these variable namespaces.

| Account-Related Namespace | Description |
|---|---|
| `waveset.accounts` | Used internally for difference detection during check-in operations. It contains the starting values for all account attributes. Do not modify this value. |
| `accountInfo.accounts` | Derived read-only information about the accounts that are linked to the user and their associated resources. Use this attribute in forms, but do not modify. |
| `accounts` | Stores the read/write copies of the account attributes. Updatable fields should point to this namespace. |
| `global` | Stores copies of global attributes. Values in this area appear only if the form defines global fields, or if you are using the special MissingFields reference. (The form determines how global attributes are processed.)<br>If you set a global attribute in a workflow, you must also define a global field in the form. Simply depositing a global value in the view is insufficient. |

Table 3. Account-Related User View Attributes

## Referencing Attributes

Within a form, you can reference attributes in two ways:

- Use the name attribute of a `Field` element by adding the complete attribute pathname as follows:

  ```
  <Field name='waveset.accountId'>
  ```

  For more information on setting the `Field` name element in a form field, see the chapter titled *Identity Manager Forms*.

- Reference an attribute from within another field:

  ```
  <Expansion>
     <concat>
         <ref>global.firstname</ref><s> </s>
        <ref>global.lastname</ref>
     </concat>
  </Expansion>
  ```

Within workflow, you can reference `Field` attributes as process variables (that is, variables that are visible to the workflow engine) or in XPRESS statements for actions and transitions. When referencing these attributes in workflow, you must prefix the path with the name of the workflow variable where the view is stored (for example, ***user***`.waveset.accountId`).

## Attributes with Transient Values

You can define fields that store values at the top-level of the user view, but these values are transient. Although they exist throughout the life of the in-memory user view (typically the life of the process), the values of these fields are not stored in the Identity Manager repository or propagated to a resource account.

For example, a phone number value is the result of concatenating the values of three form fields. In the following example, `p1` refers to the area code, `p2` and `p3` refer to the rest of the phone number. These are then combined by a field named `global.workPhone`. Because the combined phone number is the only value you want propagated to the resources, only that field is prepended with `global`.

In general, use the top-level field syntax if you are:

- not pushing a field value out to Identity Manager or any other resource
- the field is being used only in email notifications or for calculating other fields.

Any field that is to be passed to the next level must have one of the path prefixes defined in the preceding table, User View Attributes.

```
<Field name='p1' required='true'>
   <Display class='Text'>
      <Property name='title' value='Work Phone Number'/>
      <Property name='size' value='3'/>
      <Property name='maxLength' value='3'/>
   </Display>
</Field>
<Field name='p2' display='true' required='true'>
   <Display class='Text'>
      <Property name='rowHold' value='true'/>
      <Property name='noNewRow' value='true'/>
      <Property name='size' value='3'/>
      <Property name='maxLength' value='3'/>
   </Display>
</Field>
<Field name='p3' display='true' required='true'>
   <Display class='Text'>
      <Property name='rowHold' value='true'/>
      <Property name='noNewRow' value='true'/>
      <Property name='size' value='4'/>
```

```
      <Property name='maxLength' value='4'/>
   </Display>
</Field>
<Field name='global.workPhone' required='true' hidden='true'>
   <Expansion>
      <concat>
         <ref>p1</ref>
         <s>-</s>
         <ref>p2</ref>
         <s>-</s>
         <ref>p3</ref>
      </concat>
   </Expansion>
</Field>
```

## waveset Attribute

The `waveset` attribute set contains the information that is stored in a `WSUser` object in the Identity Manager repository. Some attributes nested within this attribute set are not intended for direct manipulation in the form but are provided so that Identity Manager can fully represent all information in the `WSUser` object in the view.

### Most Commonly Used Attributes

Not all attributes are necessary when creating a new user. The following list contains the `waveset` attributes that are most often visible during creation or editing. Some attributes are read-only, but their values are used when calculating the values of other attributes. All `waveset` attributes are described in the sections that follow this table.

| Attribute | Editable? | Data type |
|---|---|---|
| waveset.accountId | Read/Write | String |
| waveset.applications | Read/Write | String |
| waveset.correlationKey | Read/Write | String |
| waveset.creator | Read only | String |
| waveset.createDate | Read only | String |
| waveset.disabled | Read/Write | String |
| waveset.email | Read/Write | String |
| waveset.exclusions | Read/Write | List |
| waveset.id | Read | String |
| waveset.lastModDate | Read | String |
| waveset.lastModifier | Read | String |
| waveset.locked | Read | String |
| waveset.lockExpiry | Read/Write | String |
| waveset.organization | Read/Write | String |
| waveset.questions | Read/Write | List |
| waveset.resources | Read/Write | List |
| waveset.roles | Read/Write | String |

Table 4. Most Commonly Used Attributes of the waveset Attribute (User View)

## waveset.accountId

Specifies the visible name of the Identity Manager user object. It must be set during user creation. Once the user has been created, modifications to this attribute will trigger the renaming of the Identity Manager account.

For information on renaming a user, see *Identity Manager Administration*.

## waveset.applications

Contains a list of the names of each application (also called *resource group* in the Identity Manager user interface) assigned directly to the user. This does not include applications that are assigned to a user through a role.

## waveset.attributes

Collection of arbitrary attributes that is stored with the `WSUser` in the Identity Manager repository. The value of the `waveset.attributes` attribute is either null or another object. The names of the attributes in this object are defined by a system configuration object named *Extended User Attributes*. Common examples of extended attributes are `firstname`, `lastname`, and `fullname`. You can reference these attributes in the following ways:

```
waveset.attributes.fullname
```

or

```
accounts[Lighthouse].fullname
```

**Note** You typically do not modify the contents of the `waveset.attributes` attribute. Instead, modify the values of the `accounts[Lighthouse]` attributes. When the attribute is stored, values in `accounts[Lighthouse]` are copied into `waveset.attributes` before storage. `waveset.attributes` is used to record the original values of the attributes. The system compares the values here to the ones in `accounts[Lighthouse]` to generate an update summary report. See the section on the `account[Lighthouse]` attribute for an example of how to extend the extended user attributes.

## waveset.correlationKey

Contains the correlation value used to identify a user during reconciliation and discovery of users. You can directly edit it, although it is generally not exposed.

## waveset.creator

Contains the name of the administrator that created this user.

This attribute is read-only.

## waveset.createDate

Contains the date on which this account was created. Dates are rendered in the following format: `MM/dd/yy HH:mm:ss z`

### Example

```
05/21/02 14:34:30 CST
```

This attribute is set once only and is read-only.

## waveset.disabled

Contains the disabled status of the Identity Manager user. It is set to a value that is logically true if the account is disabled. In the memory model, it is either a Boolean object or the string `true` or `false`. When accessed through forms, you can assume it is a string.

You can modify this attribute to enable or disable the Identity Manager user, although it is more common to use the `global.disable`. (Prepending `global.` to a variable name ensures that the system applies the value of that variable to all resources that recognize the variable, including Identity Manager.)

Once this value becomes true, the user cannot log in to the Identity Manager user interface.

## waveset.email

Specifies the email address stored for a user in the Identity Manager repository. Typically, it is the same email address that is propagated to the resource accounts.

Modifications to this attribute apply to the Identity Manager repository only. If you want to synchronize email values across resources, you must use the `global.email` attribute.

You can modify this attribute.

## waveset.exclusions

List the names of the resource that will be excluded from provisioning, even if the resource is assigned to the user through a role, resource group, or directly.

## waveset.id

Identifies the repository ID of the Identity Manager user object. Once the user has been created in Identity Manager, this value is non-null. You can test this value to see if the user is being created or edited. This attribute is tested with logic in the form. You can use it to customize the displayed fields depending on whether a new user is being created (`waveset.id` is null) or an existing user account is being edited (`waveset.id` is non-null).

### Example

The following example shows an XPRESS statement that tests to see if `waveset.id` is null:

```
<isnull><ref>waveset.id</ref></isnull>
```

## waveset.lastModDate

Contains the date at which the last modification was made. It represents the date by the number of milliseconds since midnight, January 1970 GMT. This attribute is updated each time a user account is modified.

This attribute is read-only.

## waveset.lastModifier

Contains the name of the administrator or user that last modified this user account.

This attribute is read-only.

## waveset.lock

Indicates whether the user is locked. A value of `true` indicates that the user is locked.

## waveset.lockExpiry

Specifies when the user lock expires if the user's Lighthouse Account policy contains a non-zero value for the locked account expiry date. This attribute value is a human-readable date and time.

## waveset.organization

Contains the name of the organization (or `ObjectGroup`) in which a user resides. An administrator can modify this attribute if he has sufficient privileges for the new organization.

Since changing an organization is a significant event, the original value of the organization is also stored in the `waveset.original` attribute, which can be used for later comparison.

## waveset.original

Contains information about the original values of several important attributes in the `waveset` attribute. The system sets this value when the view is constructed and should never be modified. The system uses this information to construct summary reports and audit log records.

Not all of the original `waveset` attributes are saved here. The attributes currently defined for change tracking are:

- `password`
- `role`
- `organization`

To reference these attributes, prepend `waveset.original.` to the attribute name (for example, `waveset.original.role`).

## password

Specifies the Identity Manager user password. When the view is first constructed, this attribute does not contain the decrypted user password. Instead, it contains a randomly generated string.

The `password` attribute set contains the attributes described in the following table.

| Attribute | Description |
|---|---|
| password | Identifies the password to be set |
| confirmPassword | Confirms the password to be set. The password should match the value of password.password |
| targets | Specifies a list of resources that can have their password changed |
| selectAll | Specifies a Boolean flag that signifies that the password should be pushed to all of the resources |
| accounts | Specifies a list of objects that contains information about each of the resources. This attribute contains two attributes, which are described below. |
| accounts.selected | Boolean. When set, indicates that the password should be changed on the resource. |
| accounts.expire | Boolean. When set, indicates that the password will expire. |

Table 5. Attributes of the password Attribute (User View)

## waveset.passwordExpiry

Contains the date on which the Identity Manager password will expire. When the view is initially constructed, the memory representation will be a java.util.Date object. As the view is processed with the form, the value can either be a Date object or a String object that contains a text representation of the date in the format mm/dd/yy.

## waveset.passwordExpiryWarning

Contains the date on which warning messages will start being displayed whenever the user logs into the Identity Manager User Interface. This is typically a date prior to the waveset.passwordExpiry date in the same format (mm/dd/yy).

## waveset.questions

Contains information about the authentication questions and answers assigned to this user. The value of the attribute is a List whose elements are waveset.questions attributes.

The waveset.questions attribute set contains the attributes described in the following table.

| Attribute | Editable? | Description |
|---|---|---|
| answer | Read/Write | Encrypted answer to the question |
| id | Read | System-generated ID for the question |
| name | Read | Name used to identify this question |
| question | Read | Text of the authentication question |

Table 6. waveset.questions Attributes (User View)

The name attribute is not stored. The system generates the name by transforming the id. This is necessary because question IDs are typically numbers, and numbers that are used to index an array in a path expression are considered absolute indexes rather than object names.

For example, the path waveset.questions[#1].question addresses the second element of the questions list (list indexes start from zero). However, since there may be only one question on the list whose ID is the number 1, the ID is not necessarily suitable as a list index. To reliably address the elements of the list, the system manufactures a name for each question that consists of the letter Q followed by the ID (in this example, Q1). The path waveset.questions[Q1].question then always correctly addresses the question.

## waveset.resources

Contains a list of the names of each resource assigned directly to the user. This does not include resources that are assigned to a user through a role or through applications. For a way to find all resources that are assigned to a user, see the section on the accountInfo attribute.

## waveset.roles

Contains the names of the roles assigned to this user. An administrator can modify this attribute if he has sufficient privileges for the new roles.

Since changing a role is a significant event, the original value of the role attribute is also stored in the original view, which can be used for later comparison.

# accounts Attribute

The `accounts` attribute contains a list of objects for each account linked to the Identity Manager user. Each account object contains the values of the account attributes retrieved from the resource.

The name of each account object is typically the name of the associated resource. If more than one account exists for a given resource, the object names take a suffix of the form `|n` where ***n*** is an integer. The first account on a resource has no suffix. The second account has the suffix `|2`. The third account on a resource has `|3`, etc.

For example, if you have a resource named Exchange Server that defines an account attribute named Profile, the view path to this attribute would be:

```
accounts[Exchange Server].Profile
```

If this view path were used in a form field, it would prevent the value of the `global.Profile` attribute from being propagated to the Exchange Server account.

**Note** You may want to use account-specific attributes in forms rather than global attributes to prevent propagation of values to all resources

## Overriding Resource Attributes

In addition to setting account attributes, you can also specify *resource attribute overrides* for each account. Resource attributes are attributes that are defined for the resource definition in Identity Manager, and consequently for the resource type. They are not attributes associated with an individual account. Examples of resource attributes include the host name of the server, or the base context in a directory.

You may want to create an account on a resource, but use a different value for one of the resource attributes. You could do this by duplicating the resource and changing the value, but excessive resource duplication can be confusing. Instead, resource attributes can be overridden on a per-account basis in the view.

Resource attribute overrides are stored in the attribute object under an attribute named `resourceAttributes`. If, for example, the resource defined an attribute named `host`, this could be specified in the view with the path:

```
accounts[Exchange Server].resourceAttributes.host
```

**Note** Although overriding resource attributes is not recommended, sometimes you cannot avoid it. You might choose to overwrite a resource to avoid creating duplicate resources that point to the same physical resource but differ by one attribute. For example, in a customer environment that has multiple Exchange 5.5 servers, it may make more sense to override the resource attribute Exchange Server in the form than to create a new resource. Contact your Identity Manager support representative for more information.

## accounts[Lighthouse]

Sets the values of only the attributes stored in the Identity Manager repository. When a view is created, it contains a copy of the attributes in the `waveset.attributes` attribute set. When the view is saved, the system compares the contents of `accounts[Lighthouse]` with `waveset.attributes` to generate and update reports and audit log entries. Although this attribute is stored in the Identity Manager repository, changes to this attribute are not automatically propagated to resources.

The *Extended User Attributes* Configuration object defines the attributes that are allowed in this view. The system ignores any name found in this set of attributes that is not registered in the configuration object.

The following code is a sample of the *Extended User Attributes* Configuration object. This object maintains the list of attributes that are managed by the `waveset.attribute` set.

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE Configuration PUBLIC 'waveset.dtd' 'waveset.dtd'>
<!--  id="#ID#Configuration:UserExtendedAttributes" name="User
Extended Attributes"-->
 <Configuration id='#ID#Configuration:UserExtendedAttributes'
name='User Extended Attributes' creator='Configurator'
createDate='1019603369733' lastMod='2' counter='0'>
  <Extension>
    <List>
      <String>firstname</String>
      <String>lastname</String>
      <String>fullname</String>
<!—add string values here - - >
      <String>SSN</String>
    </List>
  </Extension>
```

```
  <MemberObjectGroups>
    <ObjectRef type='ObjectGroup' id='#ID#Top' name='Top'/>
  </MemberObjectGroups>
</Configuration>
```

This object can be modified to extend the list from the default `firstname`, `lastname`, and `fullname` attributes. In this case, an attribute called `SSN` has been added.

### accounts[Lighthouse].properties

The value of this attribute is an object whose attribute names correspond to the properties defined by the user. User properties allow arbitrary custom data to be stored with the user in the Identity Manager repository. Properties can then be used in forms and workflows. A property is similar in some ways to an Extended User Attribute, but are not limited to primitive data types such as strings or integers.

Identity Manager defines the `tasks` system property, which is used by the Deferred Task Scanner to cause workflow tasks to be run at some date in the future. The value of the `tasks` property is a list of objects. The following table defines the attributes that belong to objects in the list.

| Attribute | Description |
|---|---|
| `name` | Identifies the name of the TaskDefinition object to run. |
| `date` | Specifies the date on which to run the task. |
| `taskName` | Identifies the TaskInstance that is created. If none is specified, Identity Manager generates a random name. |
| `owner` | Identifies the name of an Identity Manager administrator that is considered to be the owner of the task. If none is specified, the default owner is Configurator. |
| `organization` | Identifies the Identity Manager organization that the TaskInstance will be placed in. If none is specified, an organization controlled by the task owner is selected at random. |
| `description` | Descriptive text that will be stored in the TaskInstance when it is created. This text is displayed in the task status page of the Identity Manager Administrator Interface. |

## Sample Use

You can use the `accounts[Lighthouse].properties` value to display a table of the deferred tasks assigned to a user. This list is added to the form library named Default User Library, which is found in sample/formlib.xml.

The field that displays the deferred task table is named Deferred Tasks. After modifying the waveset.properties attribute, the deferred task table is now referenced by the default Tabbed User Form. If any deferred tasks exist, the table will be displayed at the bottom of the Identity tab panel.

### accounts[Lighthouse].viewUserForm

Used to display a view-only User form. This view-only form displays field information as Labels, to ensure that the administrator cannot change values, although he can list, view, and search on this user information. (The administrator selects a user from the accounts list, then clicks **View** to see user details.)

## accounts[<resource>].properties

Used to store account properties in the Identity Manager repository. Use this attribute if you have some information about the account -- for example the date it was created -- that cannot be stored as a native account attribute on the resource.

## global Attribute

You can use the global attribute set of the user view to conveniently assign attributes to many resource accounts (including Identity Manager). The value of the `global` attribute is an object whose attributes are referred to as *global attributes*. When the view is saved, the system assigns the value of each global attribute to all resource accounts that define the global attribute name in their schema map. These values are also propagated to the Identity Manager repository if there is an extended attribute with the same name.

For example, two resources *R1* and *R2* define an attribute named `fullname`. When the attribute `global.fullname` is stored in the view, this value is automatically copied into attributes `accounts[R1].fullname and accounts[R2].fullname`.

You can also use global attributes to assign extended attributes that are stored in the Identity Manager repository. If a global attribute is also declared as an extended Identity Manager attribute, it is copied into `accounts[Lighthouse]`.

**Note** Do not use `global.accountId` when creating accounts. The account ID is created by the DN templates on the resources. Using `global.accountId` overrides this, which may cause problems.

## Referencing Two Different Fullname Attributes

The `global` attribute can be used in combination with the `account` attribute for the same attribute name. For example, on an Active Directory resource, the structure of the `fullname` is `lastname, firstname`. But all other resources that have a `fullname` use `firstname lastname`.

The following example shows how you can reference these two fields in a form.

```
<Field name='global.fullname'>
    <Expansion>
       <concat>
         <ref>global.firstname</ref><s> </s>
         <ref>global.lastname</ref>
       </concat>
    </Expansion>
</Field>

<Field name='accounts[ActiveDir].fullname'>
     <Expansion>
       <concat>
          <ref>global.lastname</ref><s>, </s>
          <ref>global.firstname</ref>
      </concat>
   </Expansion>
</Field>
```

In the preceding example, creating a new user works as expected. However, when you load the user, the `fullname` attribute from the Active Directory resource can be used to populate the `global.fullname` field.

A more accurate implementation for this scenario would be to declare one resource to be the authoritative source for an attribute and create a `Derivation` rule such as the following:

```
<Field name='global.fullname'>
   <Derivation>
       <or>
         <ref>accounts[LDAP res].fullname</ref>
         <ref>accounts[NT res].fullname</ref>
       </or>
   </Derivation>
   <Expansion>
      <concat>
          <ref>global.firstname</ref><s> </s>
          <ref>global.lastname</ref>
      </concat>
   </Expansion>
</Field>
```

By defining a `Derivation` rule, the value of the `fullname` attribute in the LDAP resource will be used first to populate the `fullname` field. If the value does not exist on LDAP, then the value will be set from the NT resource.

## accountInfo Attribute

Contains read-only information about resource accounts associated with the user. It is used within system views besides the user view. Some information in this view is a duplicate of the information found in the `waveset.accounts` attribute. There are two reasons for this duplication:

- Information in this view is structured so that it is easier to use in forms
- This view can be used as a component of other views without including the entire `waveset` view.

Most account information is stored in the `accountsInfo.accounts` attribute. Other attributes simply contain lists of account names. It is common to use a `FieldLoop` in a form to iterate over the names in one of the name list attributes, then use this name to index the account list attribute.

For example, the following form element generates a list of labels that contain the names of each resource that is assigned indirectly through a role.

```
<FieldLoop for='name' in='accountInfo.fromRole'>
   <Field name='accountInfo.accounts[$(name)].name>
      <Display class='Label'/>
   </Field>
</FieldLoop>
```

The following tables shows the `accountInfo` view attributes, which describe characteristics about the user.

| Attribute | Description |
|---|---|
| `accountInfo.accounts` | Lists objects that contain information about each resource account associated with the user (for example, `created, disabled`). |
| `accountInfo.assigned` | Lists the resources that are assigned to the user. |
| `accountInfo.fromRole` | Lists (in flat list format) resources assigned to the user through the role. |
| `accountInfo.privates` | Lists (in flat list format) resources assigned directly to the user. |
| `accountInfo.toCreate` | Lists names of all resources currently assigned to the user but for which accounts do not yet exist in Identity Manager. |
| `accountInfo.toDelete` | Lists names of resources that are no longer assigned to the user, but that are still known to exist. |
| `accountInfo.types` | Lists each type of resource that is currently assigned to the user or through Reserve Groups. |
| `accountInfo.typeNames` | Lists unique type names for every assigned resource. |

Table 7. accountInfo Attributes (User View)

## accountInfo.accounts

Contains a list of objects that themselves contain information about each associated resource account. Elements in the accounts list are referenced by name, where the name is the name of the resource.

### Example

```
accountInfo.accounts[Microsoft Exchange].type
```

Objects found in the `accountInfo.accounts` list have the following attributes, as defined in the following table.

| Attribute | Description |
|---|---|
| attributes | Information about all the account attributes defined by this resource. |
| name | Name of the resource where the account exists or will be created. |
| id | Repository ID of the resource. |
| type | Resource type name. |
| accountId | Name of the user's account on this resource. |
| assigned | True if the account is currently assigned. Accounts that are not assigned can be deleted by Identity Manager. |
| protected | True if the account is currently protected. This means that update or delete operations on the account are ignored. |
| passwordPolicy | Information about the password policy defined for this resource. |

Table 8. accountInfo.accounts. Attributes (User View)

## accountInfo.accounts[ ].attributes[ ]

Contains information about all the account attributes defined by this resource. These attributes are listed on the schema map page of the resource. The value of the attribute is a List of objects.

The following table defines the attributes that these objects contain.

| Attribute | Description |
|---|---|
| `name` | The name of the Identity Manager resource account attribute. This name is defined in the resource schema map. |
| `syntax` | The syntax of the attribute value. The value of the `syntax` attribute is one of the following values.<br>• int<br>• Boolean<br>• encrypted |
| `multi` | True if the attribute allows multiple values. |

Table 9. accountInfo.accounts. Attributes (User View)

If you are designing a form, do not worry about the declared resource account attribute types. The user view processing system makes the appropriate type coercions when necessary.

## accountInfo.accounts[].passwordPolicy

A resource can be assigned a password policy. If an attribute has an assigned password policy, the value of this attribute will contain information about it.

The following table defines the attributes in the `accountInfo.accounts[resname].passwordPolicy`.

| Attribute | Description |
|---|---|
| `name` | The name of policy. This corresponds to the name of a `policy` object in the Identity Manager repository. |
| `summary` | A brief text description of the policy including information about each of the policy attributes. |
| `attributes` | The value of this attribute is another object that contains the names and values of each policy attribute. |

Table 10. accountInfo.accounts[resname].passwordPolicy Attributes (User VIew)

Applications that display policy information typically display the summary text, but if you need more fine-grained control over the display of each policy attribute, you can use the attributes map.

Forms that provide an interface for changing and synchronizing passwords often use this information.

### accountInfo.accounts[Lighthouse]

This special entry in the `accountInfo` list is used to hold information about the Identity Manager default password policy. This is convenient when displaying password forms since information about the Identity Manager password and policies must be displayed along with the information for resource accounts.

This element is present only when pass-through authentication is not being used. The resource type is `Lighthouse`.

## accountInfo Resource Name Lists

The `accountInfo` view includes attributes that contain lists of resource names. Each list is intended to be used in forms with `FieldLoop` constructs to iterate over resources with certain characteristics.

The `accountInfo` attributes that can contain resource names are:

- `assigned`
- `created`
- `fromRole`
- `private`
- `toCreate`
- `toDelete`

## accountInfo.assigned

Identifies the resources that are assigned to the user. If you are designing a form, you can call this attribute to display a list of resources that are assigned from the role, applications, and that are directly assigned to a user.

## accountInfo.typeNames

A list of unique type names for every assigned resource. This is commonly used in `Disable` expressions in forms where you want to disable fields unless a resource of a particular type is selected.

### Example

```
<Field name='HomeDirectory' prompt='Home Directory'>
   <Display class='Text'/>
      <Disable>
         <not>
            <contains>
               <ref>accountInfo.typeNames</ref>
               <s>Solaris</s>
            </contains>
         </not>
      </Disable>
</Field>
```

This returns the same information as the path `accountInfo.types[*].name` but is more efficient, which is important when used with `Disable` expressions. This list can include common resource types.

You can determine the resource type names by bringing up the resource list from the Identity Manager Administrator Interface. The **Type** column on this page contains the names of the type of currently defined resources. The options list next to **New Resource** also contains the names of the resource adapters that are currently installed.

## accountInfo.types

This attribute contains information about each type of resource that is currently assigned. The value of the attribute is a List (objects).

The following table shows the attributes that belong to each object.

| Attribute | Description |
|---|---|
| `accounts` | List of `accountIds` for each account assigned to the user that is of this type |
| `name` | Resource type name |

Table 11. accountInfo.types Attributes (User VIew)

For example, you can determine a list of IDs for all UNIX accounts with the following path: `accountInfo.types[Unix].accounts`

## display Attribute

The `display` attribute contains information that relates to the context in which the view is being processed. Most of the attributes are valid only during interactive form processing.

The following table shows the most commonly used `display` view attributes.

| Attribute | Description |
|---|---|
| `eventType` | Indicates whether the user view is servicing a create or update request, as indicated by the values create or update (read-only). |
| `session` | A handle to an authenticated Identity Manager session. This attribute is valid only during interactive editing session in the Identity Manager Administrator Interface. It is provided as an access point into the Identity Manager repository. The value of this attribute can be passed to methods in the `com.waveset.ui.FormUtil` class.<br>The `display.session` attribute is not valid in the following cases where form processing may occur:<br>• in the bulk loader<br>• during background reprovisioning<br>• in unsynchronized actions or approvals<br>Best practices suggest using this attribute only within a `Property` or `Constraints` element. In almost all existing forms, `display.session` is used only in `Constraints` elements. |
| `subject` | An object holding information about the credentials of an Identity Manager user or administrator. This value is set in almost all cases, but is typically used in workflow applications called during background activities where the `display.session` is no longer valid. The subject can be used to get a new session. In this case, it is used for gaining access to the repository. |
| `state` | A handle to a `_com.waveset.ui.util.RequestState_` object that in turn contains handles to objects related to the HTTP request such as the `_javax.servlet.http.HttpSession_`. |

Table 12. Most Commonly Used display Attributes (User VIew)

## Default itemType Behavior

Typically, only `wizard` itemTypes cause a workflow to transition directly to a WorkItem if the requester is the owner of the workItem.

When `itemType` is set as follows, the workflow will not transition into a WorkItem, but will instead appear under the Approval tab:

- `approval`
- `custom`
- `itemType`

### Overriding Default Behavior

You can override behavior in the User view by setting the `allowedWorkItemTransitions` option as a property of the form as follows:

```
<Form ......>
        <Properties>
          <Property name='allowedWorkItemTransitions'>
            <list>
              <s>myCustomType</s>
            </list>
          </Property>
        </Properties>
```

## Deferred Attributes

A *deferred attribute* is an attribute that derives its value from an attribute value on a different account. You declare the deferred attribute in a view (and the WSUser model), and the provisioning engine performs this substitution immediately before calling the adapter.

If the deferred attribute derives its value from another resource's GUID attribute, the source adapter does not need to take action. However, if the source attribute is not the GUID, the adapter must return the attribute in the ResourceInfo._resultsAttributes map as a side effect of the realCreate operation. If the adapter does not return the attribute, the provisioning engine will fetch the account to get the value. This is less efficient than modifying the adapter to return the value.

### When to Use Deferred Attributes

Use deferred attributes when creating new accounts to specify that the value of an account attribute is to be derived from the value of an attribute on a different account that will not be known until the source account has been created. One common example is to set an attribute to the value of the generated unique identifier.

### Using Deferred Attributes

There are two main steps to defining a deferred attribute:

1. Ensure that the account is created on the source resource before the second account is created. Do this by creating an ordered Resource Group that contains both resources and assigning the Resource Group to the user.
2. Set the special attributes in the User view for the accounts that are to be created as indicated by the following sample scenario. Each deferred attribute requires two view attributes: one that identifies the source account, and one that identifies the source attribute. Set these using paths of the following form:

   ```
   accounts[<resource>].deferredAttributes.<attname>.resource
   accounts[<resource>].deferredAttributes.<attname>.attribute
   ```

   where `<resource>` would be replaced with an actual resource name and `<attname>` replaced with an actual attribute name.

   For example, assume a scenario in which the following two resources are created: 1) a resource named LDAP that generates a `uid` attribute when an account is created; 2)a resource named HR, which contains a `directoryid` attribute named `directoryid`, whose value is to be the same as `uid` in the LDAP resource.

   The following form fields set the necessary view attributes to define this association.

   ```
   <Field
   name='accounts[HR].deferredAttributes.directoryid.resource'>
      <Expansion><s>LDAP</s></Expansion>
   </Field>
   <Field name='accounts[HR].deferredAttributes.directoryid
      <Expansion><s>uid</s></Expansion>
   </Field>
   ```

## Debugging the User View

When debugging the User view, you might find it useful to dump the contents of the view into a new file. To create a dump file, add the following Derivation statement to the User view:

```
<Field name='DumpView'>
   <Derivation>
      <invoke name='dumpFile'>
         <ref>form_inputs</ref>
            <s>c:/temp/view.xml</s>
      </invoke>
   </Derivation>
</Field>
```

This Derivation expression invokes the `dumpFile` method, which generates the file after the User form is displayed for the first time. The `form_inputs` variable is automatically bound to the view that is being used with this form.

In the preceding example, the String argument to the `dumpFile` method is a file system path, where you substitute a valid path for `c:/temp/view.xml`.

# Account Correlation View

Used to search for users correlating to a specified account (or account attributes). This view is used as part of the account reconciliation process.

This view contains the root attributes listed below. The values of these attributes are GenericObjects. The new ID is <*account_name*>@<*resource_name*>

| Attribute | Description |
|---|---|
| correlation | Contains information about how correlation should be done |
| matches | Contains the result of the correlation |

The correlation request is executed on both the view get operation and refresh request. In the case of a refresh, the request specified in the view is used (with the exception of accountId and resource, as these values are overridden by the view ID). In the case of a get request, view options of the same name as the view attribute (for example, correlator) can be used to specify the view-supplied portion of the request.

**Note** accountAttributes, when provided as a view option, can be supplied as a WSUser (as returned by resource adapter methods) or as a GenericObject.

## Correlation

| Attribute | Editable? | Data Type | Required? |
|---|---|---|---|
| accountId | Read | String | Yes |
| accountGUID | Read/Write | String | No (unless if accountId and resource cannot clearly identify the resource) |
| resource | Read | String | Yes |

| Attribute | Editable? | Data Type | Required? |
|---|---|---|---|
| accountAttributes | Read/Write | String | |
| correlator | Read/Write | String | No |
| confirmer | Read/Write | String | No |

## accountId

Specifies the name of the account to correlate. This is automatically obtained from the view ID.

## accountGUID

Specifies the GUID of the account to correlate. Required only if `accountId` and `resource` cannot clearly and unambiguously identify the resource.

## resource

Specifies the name of the resource where the account resides. This value is automatically obtained from the view ID.

## accountAttributes

Specifies the attributes of the account. If present, the viewer will not fetch the current account attributes to pass to the correlation/confirmation rules. Instead, these attributes will be passed in.

## correlator

Specifies the correlation rule to use. If not present, the correlation rule specified by reconciliation policy for the resource will be used. If present, but null, no correlation rule is used.

## confirmer

Specifies the confirmation rule to use. If not present, the confirmation rule specified by reconciliation policy for the resource will be used. If present, but null, no confirmation rule is used.

These lists consist of GenericObjects that contain the summary attributes of users.

| Attribute | Editable? | Data Type |
|---|---|---|
| claimants | Read | List |
| correlated | Read | List |
| unconfirmed | Read | List |

## claimant

Lists claimants that are calculated independent of the correlation algorithm, so claimants may also appear in another of the lists. Claimant discovery can be disabled by setting `ignoreClaimants` to true in the view options. A user claims an account if it has a `ResourceInfo` explicitly referencing the account.

## correlated

Lists the users who were correlated to the resource account.

## unconfirmed

Lists users who were selected by the correlation rule, but were rejected by the confirmation rule. This list is only present if the `includeUnconfirmed` is set to `true` in the view options.

# Admin Role View

Used when creating or updating an admin role to a user. *Admin roles* enable you to define a unique set of capabilities for each set of organizations. Capabilities and controlled organizations can be assigned directly or indirectly through roles.

One or more admin roles can be assigned to a single user and one or more users can be assigned the same admin role.

| Name | Editable? | Type | Required? |
|---|---|---|---|
| id | Read/Write | String | No |
| name | Read/Write | String | Yes |
| capabilities | | List | Yes |
| capabilitiesRule | | String | Yes |
| controlledOrganizations | | List | Yes |
| controlledOrganizationsRule | | String | Yes |
| controlledOrganizationsUserform | | String | Yes |
| controlledSubOrganizations | | List (object) | No |
| memberObjectGroup | | List | Yes |

Figure 4. Admin Role View Attributes

## id

Uniquely identifies the AdminRole object in Identity Manager. System-generated.

## name

Specifies the name of the admin role.

## capabilities

Identifies the list of capability names that are assigned to this admin role.

## capabilitiesRule

Specifies the name of the rule to be evaluated that will return a list of zero or more capability names to be assigned.

## controlledOrganizations

Lists organization names over which the associated capabilities are allowed.

## controlledOrganizationsRule

Specifies the name of the rule to be evaluated. This rule will return a list of zero of more controlled organizations names to be assigned.

## controlledOrganizationsUserform

Specifies the userform that will be used when editing or creating users in the scope of organizations controlled by this admin role. Valid if the userform is not directly assigned to the user that is assigned this Admin role.

## controlledSubOrganizations

Lists the controlled organizations for which a subset of the objects available has been either included or excluded. The value of this attribute consists of a list of `controlledSubOrganization` objects. Each ControlledOrganization object view is as follows.

| Attribute | Data Type | Required? |
|---|---|---|
| `name` | String (name of controlled object group) | |
| `types` | List (objects) | |

Figure 5. controlledSubOrganizations View Attributes (Admin Role view)

`types` is a list of objects, where the list of objects to include or exclude are organized by type (for example, Resource, Role, and Policy). The view for each object type is as follows:

| Attribute | Data Type | Required? |
|---|---|---|
| name | String | |
| include | List (objects) | |
| exclude | List (objects) | |

Figure 6. controlledSubOrganizations View Attribute Object Types (Admin Role view)

### name

Specifies the name of the object type.

### include

Lists object names of the associated object type to include.

### exclude

Lists object names of the associated type to exclude.

## memberObjectGroup

Lists the ObjectGroups of which this Admin role is a member. These are the object groups (organizations) that this Admin role is available to.

# Change User Answers View

Used to change an existing user's authentication answers for one or more login interfaces.

Contains two high-level attributes.

| Attribute | Editable? | Data Type | Required? |
|---|---|---|---|
| questions | | List | |
| loginInterface | | String | |

Figure 7. Change User Answers View Attributes

## questions

Describes the question. Contains the following attributes:

| Attribute | Data Type | Required? |
|---|---|---|
| qid | String | |
| question | String | |
| answer | String | |
| answerObfuscated | Boolean | |

Figure 8. questions Attributes (Change User Answers View)

### qid

Uniquely identifies a question that is used to associate this question with one defined in the policy.

### question

Specifies the question string as defined in the policy.

### answer

Specifies the user's answer, if specified, associated with the `qid`.

### answerObfuscated

Specifies whether the answer is displayed or encrypted.

## loginInterface

Identifies the login interface with which this question is associated. Its value is a unique message catalog key for each login interface.

Contains the following attributes:

| Attribute | Data Type | Required? |
|---|---|---|
| `name` | String | |
| `questionPolicy` | String | |
| `questionCount` | String | |

Figure 9. loginInterface Attributes (Change User Answers View)

### name

Identifies the name of the login interface that the question is associated with.

Valid values include:

- UI_LOGIN_CONFIG_DISPLAY_NAME_ALL_INTERFACES
- UI_LOGIN_CONFIG_DISPLAY_NAME_ADMIN_INTERFACE
- UI_LOGIN_CONFIG_DISPLAY_NAME_BPE
- UI_LOGIN_CONFIG_DISPLAY_NAME_CLI_INTERFACE
- UI_LOGIN_CONFIG_DISPLAY_NAME_DEFAULT_USER_INTERFACE
- UI_LOGIN_CONFIG_DISPLAY_NAME_IVR_INTERFACE
- UI_LOGIN_CONFIG_DISPLAY_NAME_QUESTION_INTERFACE
- UI_LOGIN_CONFIG_DISPLAY_NAME_USER_INTERFACE

## questionPolicy

Specifies the policy that this question is associated with (for example, All, Random, Any, or RoundRobin).

## questionCount

Set only if the `questionPolicy` attribute is set to Any or Random.

# Change User Capabilities View

Used to change an Identity Manager user's capabilities.

| Attribute | Editable? | Data Type | Required |
|---|---|---|---|
| adminRoles | | List [String] | |
| capabilities | | List [String] | |
| controlledOrganizations | | List [String] | |

Figure 10. Change User Capabilities View Attributes

## adminRoles

Lists the Admin roles that are assigned to the user.

## capabilities

Lists capabilities assigned to this user.

## controlledOrganizations

Lists the organizations that this user controls with the assigned capabilities.

# Deprovision View

Used to present and select a list of resources to be deprovisioned. Contains one single top-level attribute.

## resourceAccounts

This attribute contain the following attributes.

| Name | Editable? | Data Type | Required? |
|---|---|---|---|
| id | Read/Write | String | |
| selectAll | Read/Write | Boolean | |
| unassignAll | Read/Write | Boolean | |
| unlinkAll | Read/Write | Boolean | |
| currentResourceAccounts | Read | List (objects) | |

Figure 11. resourceAccounts Attributes (Deprovision View)

### id

Specifies the unique identifier for the account.

### selectAll

Controls whether all resources are selected.

### unassignAll

Specifies that all resources should be removed from the user's list of private resources.

### unlinkAll

Specifies that all resources should be unlinked from the Identity Manager user.

## tobeCreatedResourceAccounts

Represents the accounts that are assigned to this Identity Manager user but which have not been created. Passwords cannot be unlocked on accounts that have not yet been created.

## tobeDeletedResourceAccounts

Represents the accounts that have been created but are no longer assigned to this user. Passwords cannot be changed on accounts that are going to be deleted.

All three account lists contain objects that describe the state of the account on each resource and allow you to individually select accounts

## currentResourceAccounts

Represents the set of accounts that are currently being managed by Identity Manager (including the Identity Manager account itself).

All account lists are indexed by resource name.

| Name | Editable? | Data Type |
|---|---|---|
| `selected` | Read/Write | Boolean |
| `unassign` | Read/Write | Boolean |
| `unlink` | Read/Write | Boolean |
| `name` | Read | String |
| `type` | Read | String |
| `accountId` | Read | String |
| `exists` | Read | Boolean |
| `disabled` | Read | Boolean |
| `authenticator` | Read | Boolean |
| `directlyAssigned` | Read | Boolean |

Figure 12. currentResourceAccounts Attributes (Deprovision View)

### selected

If set to `true`, indicates that for a given resource, the associated account should be deprovisioned. If the selected account is Lighthouse, the Identity Manager user and all associated resource assignments will be deleted unless they are also selected. However, the associated resource accounts will not be deleted.

### unassign

If set to `true`, indicates that the specified resource should be removed from the user's list of private resources (for example, `waveset.resources`).

### unlink

If set to `true`, indicates that the specified resource should be unlinked from the Identity Manager user (for example, remove the associated ResourceInfo object).

**Note** If `selected` or `unassign` are set to `true`, this suggests that `unlink` will also be `true`. However, the converse is not true. `unlink` can be `true` and `selected` and `unassign` can be set to `false`.

### name

Specifies the name of resource. This corresponds to the name of a `resource` object in the Identity Manager repository.

### type

Identifies the type of resource, such as Solaris. You can determine the resource type names by bringing up the resource list from the Identity Manager Administrator interface. The **Type** column on this page contains the names of the type of currently defined resources. The options list next to **New Resource** also contains the names of the resource adapters that are currently installed.

### accountId

Specifies the identity of the resource account.

### exists

Indicates whether the account already exists on the resource or not (only in `currentResourceAccounts`).

### disabled

Indicates whether the account is currently disabled or enabled (only in `currentResourceAccount`).

### authenticator

Indicates whether the account is one that the user is configured to log in.

### directlyAssigned

If `true`, indicates that the account is directly assigned to the user. A value of `false` indicates that the account is indirectly assigned by a role or application.

# Disable View

Used to disable accounts on the Identity Manager user. This view is often used in custom workflows.

## resourceAccounts

Represents the top-level attribute when accessing attributes in this view.

| Name | Editable? | Type | Required? |
|---|---|---|---|
| `id` | Read | String | |
| `selectAll` | Read | Boolean | |
| `currentResourcesAccount` | Read | String | |

### id

Identifies the Identity Manager ID of the user.

### selectAll

When set, causes all resource accounts to be disabled, including the Identity Manager account.

### currentResourceAccounts

Represents the set of accounts that are currently being managed by Identity Manager, including the Identity Manager account itself. Use the `selected` field to signify that the specific resource should be enabled.

| Name | Editable? | Type |
|---|---|---|
| name | Read | String |
| type | Read | String |
| accountId | Read | String |
| exists | Read | Boolean |
| disabled | Read | Boolean |
| selected | Read/Write | Boolean |

Table 13. resourceAccounts.currentResourceAccounts Attributes (Disable View)

# Enable View

Used to enable accounts on the Identity Manager user. This view is often used in custom workflows.

## resourceAccounts

Represents the top-level attribute when accessing attributes in this view.

| Name | Editable? | Type | Required? |
|---|---|---|---|
| `id` | Read | String | |
| `selectAll` | Read | Boolean | |
| `currentResourcesAccount` | Read | String | |

### id

Identifies the user's Identity Manager ID.

### selectAll

When set, all resource accounts will be enabled, including the Identity Manager account.

### currentResourceAccounts

Represents the set of accounts that are currently being managed by Identity Manager, including the Identity Manager account itself. Use the `selected` field to signify that the specific resource should be enabled.

| Name | Editable? | Type |
|---|---|---|
| name | Read | String |
| type | Read | String |
| accountId | Read | String |
| exists | Read | Boolean |
| disabled | Read | Boolean |
| selected | Read/Write | Boolean |

Table 14. resourceAccount.currentResourceAccounts Attributes (Enable View)

# Find Objects View

Provides a customizable, generic Identity Manager repository search interface for any object type defined in Identity Manager that has rights and is not deprecated or restricted to internal use. The Find Objects view handler provides the associated forms for specifying one or more attribute query conditions and parameters and for the display of the find results. In addition, you can use view options to specify attribute query conditions and parameters.

This view contain the following attributes.

| Name | Editable? | Type | Required? |
|---|---|---|---|
| `objectType` | Read/Write | String | Yes |
| `allowedAttrs` | Read/Write | List | No |
| `attrsToGet` | Read/Write | List | No |
| `attrConditions` | Read/Write | List | No |
| `maxResults` | Read/Write | String | No |
| `results` | Read | List | No |
| `sortColumn` | Read/Write | String | No |
| `selectEnable` | Read/Write | Boolean | No |

Table 15. Top-Level Attributes (Find Objects View)

## objectType

Specifies the Identity Manager repository object type to find (for example, Role, User, or Resource).

## allowedAttrs

Lists the specified object types (specified by the `objectType` attribute) allowed queryable attribute names that are obtained by default by calling the objectType's `listQueryableAttributeAttrs()` method. This method is exposed by each class that extends PersistentObject. If not overridden by the object type class, it inherits the PersistentObject implementation returning the default set of queryable attributes supported by all PersistentObjects.

You can override the default set by specifying the set of `allowedAttrs` in either the default section or the `objectType`-specific section of the findObjectsDefaults.xml configuration file. This file resides in the sample directory. Specify each allowed attribute in the sample/findObjectsDefaults.xml file as follows:

## name

Identifies the attribute.

## displayName

Specifies the attribute name as it is displayed in the Identity Manager Administrator interface. If not specified, the value of this attribute defaults to the same value as `name`.

## syntax

Indicates the data type of attribute value where supported values include `string`, `int`, and `boolean`. If not specified, this value defaults to `string`.

## multiValued

Indicates whether the attribute supports multiple values. A value of `true` indicates that attribute supports multiple values. If unspecified, this value defaults to `false`. This attribute applies only if the attribute syntax is `string`.

## allowedValuesType

Specifies the name of the Identity Manager type if the allowed values of the attribute are instances of an Identity Manager type (for example, Role or Resource). If not specified, this attribute defaults to null.

If the `name` attribute is an Identity Manager-defined attribute, then only `name` is required. If the attribute `name` is an extended attribute, you must specify at least the `name` and, optionally, the other attributes unless the defaults are sufficient.

See `sample/findObjectsDefaults.xml` for example formats for specification of allowed attributes.

You can specify the list of `allowedAttrs` as either a list of strings, a list of objects, or a combination of both.

# attrsToGet

Lists the summary attribute names of the specified object types (`objectType`) to be returned with each object that match the specified attribute query conditions. You can obtain the object type's set of supported summary attributes by calling the object type's `listSummaryAttributeAttrs()` method. (This method is exposed by each class that extends PersistentObject.) If not overridden by the `objectType` class, it inherits the PersistentObject implementation that returns the default set of summary attributes that are supported by all Persistent Objects.

You can override the default by specifying the list of `resultColumnNames` in either the default section or the `objectType`-specific section of the sample/findObjectsDefaults.xml configuration file.

# attrConditions

Lists the attribute conditions that are used to find objects of the specified object type (`objectType`) that match the specified attribute conditions (`attrConditions`). Each attribute condition in the list should be specified as follows:

## selectedAttr

Identifies one of the attribute names from the list of allowed attributes (`allowedAttrs`).

## selectedAttrRequired

(Optional) Indicates whether the selected attribute (`selectedAttr)` can be changed for this attribute condition. A value of `true` indicates that the selected attribute cannot be changed for this attribute condition, and the attribute condition cannot be removed from the list of attribute conditions

## defaultAttr

(Optional) Identifies the `allowedAttrs` name to select by default when the list of allowed attributes is displayed in interface.

## allowedOperators

Lists the operators allowed based on the syntax specified in the selected attribute (`selectedAttr`). By default, this list is obtained by calling the `getAllowedOperators` method passing the values of the `syntax` and

multiValued attributes of the selected attribute (selectedAttr). You can override the default by specifying the set of allowed operators (allowedOperators) in either the default section or the objectType-specific section of the sample/findObjectsDefaults.xml configuration file.

## selectedOperator

Specifies the name of one operator from the list specified in allowedOperators.

## selectedOperatorRequired

(Optional) Indicates whether the selected operator (selectedOperator) can be changed for this attribute condition. A value of true indicates that the selected operator cannot be changed for this attribute condition, and the attribute condition cannot be removed from the list of attribute conditions

## defaultOperator

(Optional) Specifies the name of the operator (allowedOperators) to select by default when the list of allowed operators (allowedOperators) is displayed in the form.

## value

Indicates the value or operand for the selected attribute name and operator that must be tested when Identity Manager determines if it should return an object of the specified object type (objectType). You can omit this attribute if the value of selectedOperator is exists or notPresent.

## valueRequired

(Optional) Indicates whether the value of the attribute condition can be changed. A value of true indicates that value can be changed. It also indicates that the attribute condition cannot be removed from the list of attribute conditions.

## removeAttrCond

Determines if this attribute condition should be removed or not (internal).

You can specify attribute conditions as view options by using the FindObjects.ATTR_CONDITIONS constant or the `attrCondition` string. If `attrConditions` is not specified, Identity Manager returns all objects of the specified object type.

## maxResults

(Optional) Specifies the maximum number of objects of the specified `objectType` that Identity Manager should return from the find request. Defaults to 100 if not specified. You can override the default by specifying the a value for `resultMaxRows` attribute in either the default section or the `objectType`-specific section of the sample/findObjectsDefaults.xml configuration file.

Use of this attribute can improve performance in cases where many Identity Manager repository objects of the specified type exist.

## results

If the value of `attrsToGet` is null, the value of `result` is a list of object names that match the specified attribute condition. If the value of `attrsToGet` is non-null, `results` is a list of objects that matched the specified `attrConditions`, where each object consists of:

- ***columns*** - Lists displayable column names that match the requested `attrsToGet`
- ***rows*** - Lists ***row*** objects named from 0 to the number of rows (for example, ‘10’)
- ***row*** - Lists objects that consist of a name from ‘0’ to the number of columns (for example, ‘6’) and a value for that ***rows*** column

## sortColumn

(Optional) Indicates the value of the column to sort the results on. Defaults to '0' if not specified. You can override the default by specifying a value for `resultSortColumn` in either the default section or the `objectType`-specific section of the sample/findObjectsDefaults.xml configuration file.

## selectEnable

(Optional) Specifies whether more than one result row can be selected simultaneously. A value of `true` indicates that more than one result row can be selected. The default is `false`. The default can be overridden by specifying a value for `resultSelectEnable` in either the default section or the `objectType`-specific section of the sample/findObjectsDefaults.xml configuration file.

# Org View

Used to specify the type of organization created and options for processing it.

## Common Attributes

The high-level attributes of this view are listed in the following table.

| Name | Editable? | Data Type | Required? |
|---|---|---|---|
| `orgName` | Read | String | System-Generated |
| `orgDisplayName` | Read/Write | String | Yes (unless `orgType` equals `junction`) |
| `orgId` | Read | String | System-Generated |
| `orgParent` | Read/Write | String | Yes |
| `orgChildOrgNames` | Read | List(objects) | System-Generated |
| `orgApprovers` | Read/Write | List | No |
| `orgUserForm` | Read/Write | String | No |
| `orgViewUserForm` | Read/Write | String | No |
| `orgPolicies` | Read/Write | Object | No |
| `orgType` | Read | String | No |

Table 16. Org View Attributes

Additional attributes are valid depending on how the `orgType` field is set.

## orgName

Identifies the UID for the organization.

## orgDisplayName

Specifies the short name of the organization. This value is used for display purposes only and must be unique.

## orgId

Specifies the ID that is used to uniquely identify the organization within Identity Manager.

## orgParent

Identifies the parent organization for this organization. Null if the organization resides in the Top directory.

## orgChildOrgNames

List the names of all child organizations of this organization.

## orgApprovers

Lists the administrators who approve users added to this organization.

## orgUserForm

Specifies the `userForm` for administrators who are in this organization.

## orgViewUserForm

Specifies a read-only `userForm` for administrators who are in this organization.

## orgPolicies

Identifies policies for users in this organization. This is a map keyed by type string, with lists of policies:

- [`type`] -- Identifies the policy type
- `name` -- Specifies the name
- `id` -- Indicates the ID

- `implementation` -- Identifies the class that implements this policy

## orgType

Defines the organization type, which can the values defined in the following table.

| Value | Description | Valid Attributes |
|---|---|---|
| `none` | Creates an object group as a standard organization | `orgParent (required)`<br>`orgDisplayName (required)`<br>`orgUserMembersRule`<br>`orgUserMembersCacheTimeout` |
| `junction` | Creates an object group as a directory junction | `orgParent`<br>`orgResource` |
| `virtual` | Creates an object group as a virtual organization | `orgParent`, `orgDisplayName` |

Figure 13. Valid Values for orgType Attribute (Org View)

| Name | Editable? | Data Type | Required? |
|---|---|---|---|
| `orgUserMembersRule` | Read/Write | String | No |
| `orgUserMembersCacheTimeout` | Read/Write | Integer | No |
| `orgResource` | Read/Write | String | Yes |
| `orgContainerType` | Read/Write | String | Yes |
| `orgRefreshAllOrgs` | Read/Write | Boolean | No unless `orgAction` equals `Refresh` |

Table 17. Valid Attributes when orgType is Set (Org View)

### orgUserMembersRule

Identifies the name of rule to be evaluated to determine rule-driven user members.

### orgUserMembersCacheTimeout

Specifies the number of milliseconds before the cache times out if the user members returned by the orgUserMembersRule are to be cached. A value of 0 indicates no caching.

### orgDisplayName

Specifies the organization's display name. Must be unique within its parent organization only.

### orgContainerType

Specifies the type of directory resource container to create (for example, `o`, `ou`, `dc`). This view attribute value is required when creating a new virtual organization.

### orgRefreshAllOrgs

If `true`, synchronizes all child containers of the selected container. If `false`, syncs only child containers that are immediate children of the selected container. Defaults to `false` if not specified.

### orgRefreshAllOrgsUserMembers

If `true`, synchronizes container user membership in all child containers of the selected container. If `false`, syncs only container user membership in child containers that are immediate children of the selected container. Defaults to `false` if not specified.

### orgResource

If `junction` or `virtual` option is `true`, specifies the name or ID of the Identity Manager directory resource from which to synchronize containers and user membership.

## Using an Organizational Path Name Instead of a System-Generated ID

When calling this view in workflow, you can use either the system-generated ID or supply an organizational path expression as a value for checkoutView (for example, `top:us:central:texas`).

## Sample Workflow

```
<Activity id="1" name="Refresh Organization">
   <Variable name="orgView"/>
   <Action name="Get Organization"
        Application="com.waveset.session.WorkflowServices">
     <Argument name="op" value="checkoutView"/>
     <Argument name="subject" value="#ID#Configurator"/>
     <Argument name="viewId" value="OrgViewer:top:us:central:texas"/>
     </Argument>
      <Return from="view" to="orgView"/>
   </Action>
```

# Password View

Used by administrators to change passwords of the Identity Manager user or their resource accounts.

This view contains one top-level attribute.

## resourceAccounts

This attribute contains the following attributes.

| Attribute | Editable? | Data Type | Required? |
|---|---|---|---|
| id | Read/Write | String | Yes |
| selectAll | Read/Write | Boolean | No |
| currentResourceAccounts | Read | List (object) | No |
| tobeCreatedResourceAccounts | Read | List (object) | No |
| tobeDeletedResourceAccounts | Read | List (object) | No |
| password | Read/Write | encrypted | Yes |
| confirmPassword | Read/Write | encrypted | Yes, if view is being used interactively |

Figure 14. resourceAccounts Attributes (Password View)

### id

Specifies the account ID of the Identity Manager user whose passwords are being changed. Typically set by the view handler and never modified by the form.

### selectAll

Controls whether all password are selected.

## currentResourceAccounts

Represents the set of accounts that are currently being managed by Identity Manager (including the Identity Manager account itself).

## tobeCreatedResourceAccounts

Represents the accounts that are assigned to this Identity Manager user but which have not been created. Passwords cannot be changed on accounts that have not yet been created.

## tobeDeletedResourceAccounts

Represents the set of resources assigned to this user that are not yet being managed by Identity Manager (for example, they do not have an associated `resinfo` object). Passwords cannot be changed on accounts that are going to be deleted.

All three account lists contain objects that describe the state of the account on each resource and allow you to individually select accounts

Both resource account list are indexed by resource name, and will contain objects that describe the resources on which this user has accounts.

| Attribute | Editable? | Data Type |
|---|---|---|
| `selected` | Read/Write | Boolean |
| `name` | Read | String |
| `type` | Read | String |
| `accountId` | Read | String |
| `exists` | Read | Boolean (only in `currentResourceAccounts` |
| `disabled` | Read | Boolean (only in `currentResourceAccounts`) |
| `passwordPolicy` | Read | Object |
| `authenticator` | Read | Boolean |
| `changePasswordLocation` | Read | String (only in `currentResourceAccounts`) |
| `expirePassword` | Read/Write | Boolean |

Figure 15. tobeDeletedResourceAccounts Attributes (PasswordView)

## password

Specifies the new password you want to assign to the Identity Manager account or the resource accounts.

## confirmPassword

Confirms the password specified in the `password` attribute. When the view is used interactively, the form requires you to enter the same values in the `password` and `confirmPassword` fields. When the view is used programmatically, such as within a workflow, the `confirmPassword` attribute is ignored. If you are using this view interactively, you must set this attribute.

### selected

Indicates that the specified resource should receive the new password.

### name

Specifies the name of resource. This corresponds to the name of a `resource` object in the Identity Manager repository.

### type

Identifies the type of resource, such as `Solaris`. You can determine the resource type names by bringing up the resource list from the Identity Manager Administrator interface. The **Type** column on this page contains the names of the type of currently defined resources. The options list next to **New Resource** also contains the names of the resource adapters that are currently installed.

### accountId

Specifies the identity of the account on this resource, if one has been created.

### exists

Indicates whether the account already exists on the resource.

### disabled

Indicates whether the account is currently disabled.

### passwordPolicy

When set, describes the password policy for this resource. Can be null. It contains these attributes.

| Attribute | Description |
|---|---|
| name | String |
| summary | String |

Figure 16. passwordPolicy Attributes (PasswordView)

In addition, it contains view attributes for each of the declared policy attributes. The names of the view attributes will be the same as defined in the policy.

The summary string contains a pre-formatted description of the policy attributes.

### authenticator

If `true`, indicates that this resource is serving as the pass-through authentication resource for Identity Manager.

### changePasswordLocation

(Optional) Describes the location where the password change should occur (for example, the DNS name of a domain controller for Active Directory). The format of the value of this field can vary from resource to resource.

### expirePassword

Can be set to a non-null Boolean value to control whether the password is marked as expiring immediately after it has been changed. If null, the password expires by the default if the user whose password is being changed differs from the user that is changing the password.

## tobeCreatedResourceAccounts

Represents the accounts that are assigned to this Identity Manager user but which have not been created. Passwords cannot be changed on accounts that have not yet been created.

## tobeDeletedResourceAccounts

Represents the accounts that have been created but are no longer assigned to this user. Passwords cannot be changed on accounts that are going to be deleted.

# Process View

Used to launch tasks such as workflows or reports. The task to be launched must be defined by a TaskDefinition or TaskTemplate object in Identity Manager. Launching the task results in the creation of a TaskInstance object.

This view contains one top-level attribute named `task`. All other top-level attributes are arbitrary and are passed as inputs to the task.

## task

This top-level attribute defines how the task is to be launched.

| Attribute | Editable? | Data Type | Required? |
|---|---|---|---|
| `process` | Read/Write | String | Yes |
| `taskName` | Read/Write | String | Yes |
| `organization` | Read/Write | String | Yes |
| `taskDisplay` | Read/Write | String | No |
| `description` | Read/Write | String | No |
| `execMode` | Read/Write | String | No |
| `result` | Read/Write | WavesetResult | No |
| `owner` | Read/Write | String | No |

Figure 17. Process View Attributes

### process

Names the process to launch. This can be the name of a TaskDefinition or TaskTemplate object in Identity Manager. It can also be an abstract process name mapped through the `process` settings in the System Configuration object. This attribute is required.

### taskName

Specifies the name given to the TaskInstance object that is created to hold the runtime state of the task. If this attribute is not set, a random name is generated.

### organization

Names the organization in which to place the TaskInstance. If this attribute is not set, the TaskInstance is placed in Top.

### taskDisplay

Specifies a display name for the TaskInstance.

### description

Specifies a descriptive string for the TaskInstance. This string is displayed in the Manage Tasks table in the product interface.

### execMode

Specifies execution mode. This is typically not specified, in which case the execution mode is determined by the TaskDefinition. Setting this attribute overrides the value in the TaskDefinition.

Allowed `execMode` values are:

| Value | Description |
|---|---|
| `sync` | Specifies synchronous or foreground execution |
| `async` | Specifies asynchronous or background execution |
| `asyncImmediate` | Specifies asynchronous with immediate thread launch |

Figure 18. execMode Attribute Values (Process View)

Use the `asyncImmediate` execution mode only for special system tasks that must pass non-serializable values into the task through the view. The task thread is started immediately. The default behavior is to save the TaskInstance temporarily in the repository and have the Scheduler resume it later.

### result

Specifies the initial result for the TaskInstance. You can use this setting to pass information into the task that you eventually want displayed with the task results when the task completes.

### owner

Specifies the user name that is considered to be the owner of the task. If not set, the currently logged-in user is designated as the owner.

## View Options

The following options are recognized by the createView and checkinView methods.

### endUser

Specifies that the task is being launched from the Identity Manager User Interface. This allows users with no formal privileges to launch specially designated end-user tasks.

### process

Names the process to launch. This name is recognized by the createView method and becomes the value of the process attribute in the view.

### suppressExecuteMessage

When set to `true`, suppresses a default message that is added to the task result when an asynchronous task is launched. The default English text is, `The task is being executed in the background`.

## Checkin View Results

The following named result items can be found in the WavesetResult object that is returned by the checkinView method.

| Result | Description |
|---|---|
| `taskId` | Identifies the repository ID of the TaskInstance |
| `taskState` | Identifies the current state of the TaskInstance. It will be one of: ready, executing, suspended or finished |
| `extendedResults` | When set to `true`, indicates that the TaskInstance will have extended results. |

Figure 19. Checkin View Results

# Reconcile View

Used to request or cancel reconciliation operations on a resource. This view is used to perform on-demand reconciliation as part of a workflow. It can also be used when implementing a custom scheduler for reconciliation.

This view is write-only. get and checkout operations are not supported.

## request

Specifies the operation to perform. You must specify one of the following valid operations:

| Operation | Description |
|---|---|
| FULL | Starts a full reconciliation of the resource |
| INCREMENTAL | Starts an incremental reconciliation of the resource |
| ACCOUNT | Starts a reconciliation of the account |
| CANCEL | Cancels the currently active resource reconciliation process |

Table 18. Valid Operations for request Attribute (Reconcile View)

## accountId

Identifies the account to reconcile. This string is ignored if the request is not ACCOUNT.

## Examples

- To request a reconciliation of a single account on a resource (in this case, an Active Directory resource):

```
request = "ACCOUNT"
accountId = "cn=maurelius, ou=Austin, DC=Waveset, DC=com"
```

- To cancel the pending or currently active reconciliation process on a resource:

```
request = "CANCEL"
```

# Reconcile Policy View

Used to view and modify reconciliation policy, which is stored as part of the Identity Manager system configuration object.

## Reconciliation Policies and the Reconcile Policy View

Reconciliation policy settings are stored in a tree structure with the following general structure:

- default, or global, policy (`Default`). This is the root policy level.
- resource type (`ResType:`) policy
- resource policy (`Resource:`)

Settings can be specified at any point in the tree. If a level does not specify a value for a policy, it is inherited from the next highest policy.

The view represents an effective policy at a specified point in the policy tree, which is identified by the view name.

| View Name | Description |
|---|---|
| `Default` | Addresses the root of the policy tree |
| `ResType:`***resource type*** | Addresses the specified resource type beneath the root |
| `Resource:`***resource name*** | Addresses the specified resource beneath the resource's resource type |

Table 19. ReconcilePolicy Tree and View Names

### Policy Values

Values of policy settings are always *policy values*. Policy values can contain up to three components, as described in the following table.

| Policy Value Settings | Description |
|---|---|
| value | Specifies the value of the setting. |
| scope | Identifies the scope from which this setting is derived. Values of scope include Local, ResType, and Default, indicating which level is specifying this policy. For example, a value of SCOPE_LOCAL indicates the value is set at the current policy level.<br>SCOPE_LOCAL -- Policy is set at the resource level or current policy level<br>SCOPE_RESTYPE -- Policy is set at the restype, or resource type, level<br>SCOPE_GLOBAL. -- Policy is set at the global level |
| inheritance | Identifies the policy setting that is inherited at this level. If the scope is not Local, the inheritance will match the effective value. Not present on policy settings at the Default level. |

Table 20. Policy Value Settings Attributes (ReconcilePolicy View)

### Authorization Required

To modify the view, users require Reconcile Administrator Capability.

To access the view, users require Reconcile Administrator or Reconcile Request Administrator capabilities.

## View Attributes

The following table lists the high-level attributes of this view.

| Attribute | Description |
|---|---|
| scheduling | Contains information about automated scheduling of reconciles. |
| correlation | Contains information about how ownership of resource accounts is determined. |
| workflow | Contains information about user-supplied extensions to the reconciliation process. |
| response | Contains information about how reconciliation should respond to discovered situations. |
| resource | Contains information about how reconciliation interacts with the resource. |

Table 21. ReconcilePolicy View Attributes

## scheduling

| Attribute | Editable? | Data Type |
|---|---|---|
| reconcileServer | Read/Write | String |
| reconcileModes | Read/Write | String |
| fullSchedule | Read/Write | Schedule |
| incrementalSchedule | Read/Write | Schedule |
| nextFull | Read | Date |
| nextIncremental | Read | Date |

Table 22. scheduling Attributes (ReconcilePolicy View)

### reconcileServer

Specifies the reconciliation server that should be used to perform scheduled reconciliations.

### reconcileModes

Specifies the reconciliation modes that are enabled. Valid values are: BOTH, FULL, NONE.

### fullSchedule

Identifies the schedule for full reconciles when enabled.

### incrementalSchedule

Identifies the schedule for incremental reconciles when enabled.

### nextFull

Containing the time of the next incremental reconcile, if enabled.

### nextIncremental

Specifies the repetition count for the schedule. Schedule values are GenericObjects with the following attributes:

- count -- Specifies the repetition count for the schedule
- units -- Specifies the repetition unit for the schedule
- time -- Specifies the start time for the schedule

## correlation

Identifies the name of the correlation rule.

| Attribute | Editable? | Data Type |
|---|---|---|
| correlationRule | Read/Write | String |
| confirmationRule | Read/Write | String |

Table 23. correlation rules (ReconcilePolicy View)

### correlationRule

Identifies the name of the correlation rule to use when correlating accounts to users.

### confirmationRule

Identifies the name of the confirmation rule to use when confirming correlated users against accounts. When no confirmation is required, specify the value CONFIRMATION_RULE_NONE.

## workflow

| Attribute | Editable? | Data Type |
|---|---|---|
| proxyAdministrator | Read/Write | String |
| preReconWorkflow | Read/Write | String |
| perAccountWorkflow | Read/Write | String |
| postReconWorkflow | Read/Write | String |

Table 24. workflow Attributes (ReconcilePolicy View)

### proxyAdministrator

Specifies the name of the user with administrative capabilities.

### preReconWorkflow, perAccountWorkflow, postReconWorkflow

Specifies the name of the workflow to run at appropriate point in reconciliation processing. To specify that no workflow be run, use the value AR_WORKFLOW_NONE.

## response

| Attribute | Editable? | Data Type |
|---|---|---|
| situations | Read/Write | List |
| explanations | Read/Write | Boolean |

Table 25. response Attributes (ReconcilePolicy View)

### situations

Specifies the automated response to perform for the specified situation. Valid responses are:

| Response | Description |
|---|---|
| DO_NOTHING | Performs no automated response |
| CREATE_NEW_USER | Creates new user based on the resource account |
| LINK_ACCOUNT | Assigns the account to the claiming user |
| CREATE_ACCOUNT | Recreates the account on the resource |
| DELETE_ACCOUNT | Removes the account from the resource |
| DISABLE_ACCOUNT | Disables the account on the resource |

Table 26. situations Options (ReconcilePolicy View)

### explainActions

Specifies whether reconciliation should record detailed explanations of actions in the Account Index.

## resource

| Attribute | Editable? | Data Type |
|---|---|---|
| `reconcileNativeChanges` | Read/Write | Boolean |
| `reconciledAttributes` | Read/Write | List (of Strings) |
| `listTimeout` | Read/Write | Integer |
| `fetchTimeout` | Read/Write | Integer |

Table 27. resource Attributes (ReconcilePolicy View)

### reconcileNativeChanges

Specifies whether native changes to account attributes should be reconciled.

### reconciledAttributes

Specifies the list of account attributes that should be monitored for native changes

### listTimeout

Specifies (in milliseconds) how long reconciliation should wait for a response when enumerating the accounts present on the resource.

### fetchTimeout

Specifies (in milliseconds) how long reconciliation process should wait for a response when fetching an account from a resource.

# Reconcile Status View

Used to obtain the status of the last requested reconciliation operation. This view is read-only.

## status

Indicates the status code request (string). Valid status codes include:

| Status Code | Description |
|---|---|
| UNKNOWN | Status cannot be determined. The value of the other attribute is unspecified. |
| PENDING | Request was received, but has not been processed yet. |
| RUNNING | Request is currently being processed. |
| COMPLETE | Request has completed. Consult the attributes to determine the success or failure of the other request. |
| CANCELLED | Request was cancelled by an administrator. |

Table 28. ReconcileStatus View Attributes

## reconcileMode

Indicates the reconciliation mode of the request. Either FULL or INCREMENTAL.

## reconciler

Identifies the Identity Manager server that is processing the reconciliation request.

## requestedAt

Indicates the date on which the request was received.

## startedAt

Specifies a date on which the reconciliation operation started. If the reconciliation operation has not yet started or was cancelled while still pending, this value is null.

## finishedAt

Indicates the date on which the reconciliation operation completed. If the reconciliation process has not yet completed, this value is null.

## errors.fatal

Describes the error (if any) that terminated the reconciliation operation. Errors are returned as a list of strings.

## errors.warnings

Describes any non-fatal errors that are encountered during the reconciliation operation. Errors are returned as a list of strings.

## statistics.accounts.discovered

Identifies the number of accounts that is found on the resource at the time of the reconciliation operation.

## statistics.situation[<situation>].resulting

Identifies the number of accounts in the specified reconciliation situation after responses have been performed (successfully or not).

Valid situations are any of the following:

- CONFIRMED
- FOUND
- DELETED
- MISSING
- COLLISION
- UNMATCHED
- UNASSIGNED
- DISPUTED

# Rename User View

Used to rename the Identity Manager and resource account identities. This view is typically used when a user in a company has a name change. The other main use for this view is to change the identity of a directory user that essentially causes a move in the directory structure.

| Name | Editable? | Data Type | Required? |
|---|---|---|---|
| `newAccountId` | Read/Write | String | |
| `toRename` | Read | List | |
| `noRename` | Read | List | |
| `resourceAccounts` | Read | | |

Table 29. RenameUser View Attributes

## newAccountId

Specifies the new `accountId` to be set on the Identity Manager user and used in the Identity templates for resource accounts.

## toRename

Specifies a list of accounts in the `currentResourceAccounts` list that support the rename operation.

## noRename

Specifies a list of accounts that do not support the rename functionality.

## resourceAccounts

Contains mostly read-only information about the resource accounts. Use the following attributes to rename resource accounts:

| Attribute | Type | Description |
|---|---|---|
| selectAll | Boolean | Controls whether all accounts are renamed. |
| currentResourceAccounts [<resourcename>].selected | Boolean | Indicates that the new accountId should be used to rename the identity of this resource account. |
| currentResourceAccounts [Lighthouse].selected | Boolean | Controls whether the Identity Manager account is renamed. selectAll=true overrides this setting. |

Table 30. resourceAccounts Attributes

## accounts[<resourcename>].identity

Overrides the use of the Identity Template to create the accountId for this resource account.

## accounts[<resourcename>].<attribute>

Used when not specifying the accounts[<resourcename>].identity attribute to pass attributes to the Identity Template for the creation of the new accountId.

## Example

```
renameView.newAccountId="saurelius"
renameView.resourceAccounts.selectAll="false"
renameView.resourceAccounts.currentResourceAccounts[Lighthouse].select
ed="true"
renameView.accounts[AD].identity="cn=saurelius,OU=Austin,DC=Waveset,DC
=com"
renameView.resourceAccounts.currentResourceAccounts[AD].selected="true
"
renameView.accounts[LDAP].identity="CN=saurelius,CN=Users,DC=us,DC=com
"
renameView.resourceAccounts.currentResourceAccounts[LDAP].selected="tr
ue"
renameView.accounts[NT].identity="Marcus Aurelius"
renameView.resourceAccounts.currentResourceAccounts[NT].selected="true
"
```

# Reprovision View

Used to present and select the list of resources to be reprovisioned. This view contains one top-level attribute (resourceAccounts).

## resourceAccounts

This attribute contains the following attributes.

| Name | Editable? | Data Type | Required? |
|---|---|---|---|
| `id` | Read | String | |
| `selectAll` | Read/Write | Boolean | |
| `currentResourceAccounts` | Read | List (objects) | |

Figure 20. resourceAccounts Attributes (Reprovision View)

### id

Specifies the unique identifier for the account.

### selectAll

Controls whether all resources are selected.

### currentResourceAccounts

Represents the set of accounts that are currently being managed by Identity Manager (including the Identity Manager account itself).

All account lists are indexed by resource name.

| Name | Editable? | Data Type |
|---|---|---|
| selected | Read/Write | Boolean |
| name | Read | String |
| type | Read | String |
| accountId | Read | String |
| exists | Read | Boolean |
| disabled | Read | Boolean |
| authenticator | Read | Boolean |

Figure 21. currentResourceAccounts Attributes (Reprovision View)

### selected

If set to `true`, indicates that for a given resource, the associated account should be reprovisioned. If the selected account is Lighthouse, the Identity Manager user and all associated resource assignments will be reprovisioned unless they are also selected. However, the associated resource accounts will not be reprovisioned.

### name

Specifies the name of the resource. This corresponds to the name of a `resource` object in the Identity Manager repository.

### type

Identifies the type of resource, such as Solaris. You can determine the resource type names by bringing up the resource list from the Identity Manager Administrator interface. The **Type** column on this page contains the names of the type of currently defined resources. The options list next to **New Resource** also contains the names of the resource adapters that are currently installed.

### accountId

Specifies the identity of the resource account.

### exists

Indicates whether the account already exists on the resource or not (only in `currentResourceAccounts`).

### disabled

Indicates whether the account is currently disabled or enabled (only in `currentResourceAccount`).

### authenticator

Indicates whether the account is one that the user is configured to login.

# Reset User Password View

Used by administrators to reset a password to a randomly generated password and optionally propagate the new password to resource accounts.

## resourceAccounts

Defines characteristics of resource accounts. This attribute contains the following attributes.

| Attribute | Editable? | Data Type | Required? |
|---|---|---|---|
| id | Read | String | |
| selectAll | Read/Write | Boolean | |
| currentResourceAccounts | Read | List (object) | |
| tobeCreatedResourceAccounts | Read | List (object) | |
| tobeDeletedResourceAccounts | Read | List (object) | |

Figure 22. resourceAccounts Attributes (Reset User Password View)

### id

Specifies the account ID of the Identity Manager user whose passwords are being changed.

### selectAll

Controls whether all passwords are selected.

### currentResourceAccounts

Represents the set of accounts that are currently being managed by Identity Manager (including the Identity Manager account itself).

## tobeCreatedResourceAccounts

Represents the accounts that are assigned to this Identity Manager user but which have not been created. Passwords cannot be changed on accounts that have not yet been created.

## tobeDeletedResourceAccounts

Represents the accounts that have been created but are no longer assigned to this user. Passwords cannot be changed on accounts that are scheduled for deletion.

The three account list attributes -- `tobeDeletedResourceAccounts`, `tobeCreatedResourceAccounts`, and `currentResourceAccounts` -- contain the attributes described in the following table. These attributes describe the state of the account on each resource and allow you to individually select accounts.

| Attribute | Editable? | Data Type | Required? |
|---|---|---|---|
| `selected` | Read/Write | Boolean | |
| `name` | Read | String | |
| `type` | Read | String | |
| `accountId` | Read | String (only in `currentResourceAccounts`) | |
| `exists` | Read | Boolean (only in `currentResourceAccounts` | |
| `disabled` | Read | Boolean (only in `currentResourceAccounts`) | |
| `passwordPolicy` | Read | Object | |
| `authenticator` | Read | Boolean | |
| `changePasswordLocation` | Read | String | No |

Figure 23. tobeDeletedResourceAccounts Attributes (Reset User Password View)

### selected

Set to `true` if this account is to have its password reset.

### name

Specifies the name of resource. This corresponds to the name of a `Resource` object in the Identity Manager repository.

### type

Identifies the type of resource, such as Solaris. You can determine the resource type names by bringing up the resource list from the Identity Manager Administrator interface. The **Type** column on this page contains the names of the type of currently defined resources. The options list next to **New Resource** also contains the names of the resource adapters that are currently installed.

### accountId

Specifies the identity of the account on this resource, if one has been created.

### exists

Indicates whether the account already exists on the resource.

### disabled

Indicates whether the account is currently disabled.

### passwordPolicy

When set, describes the password policy for this resource. Can be null. It contains these attributes.

| Attribute | Data Type | Editable? | Required? |
|---|---|---|---|
| `name` | String | | |
| `summary` | String | | |

Figure 24. Reset User Password Attributes (Reset User Password View)

In addition, it contains view attributes for each of the declared policy attributes. The names of the view attributes will be the same as the WSAttribute in the Policy.

The summary string contains a pre-formatted description of the policy attributes.

### authenticator

If `true`, indicates that this resource is serving as the pass-through authentication resource for Identity Manager.

### changePasswordLocation

Describes the location where the password change should occur (for example, the DNS name of a domain controller for Active Directory). The format of the value of this field can vary from resource to resource.

# Resource View

Used when modifying resources.

| Attribute | Editable? | Data Type | Required? |
|---|---|---|---|
| name | Read/Write | String | Yes |
| adapterClassName | Read/Write | String | Yes |
| typeString | Read/Write | String | Yes |
| typeDisplayString | Read/Write | String | Yes |
| syncSource | Read/Write | | Yes |
| startupType | Read/Write | String | No |
| organizations | Read/Write | List (Strings) | Yes |
| resourceAttributes | Read/Write | List (Strings) | No |
| displayName | Read | String | No |
| type | Read | String | No |
| multivalued | Read | String | No |
| syncSource | Read/Write | Boolean | No |
| facets | Read | String | No |
| description | Read | String | No |
| noTrim | Read | String | No |
| accountAttributes | Read/Write | List (Strings) | No |
| identityTemplate | Read/Write | String | No |
| approvers | Read/Write | List (Strings) | No |
| allowedApprovers | Read | List (Strings) | No |
| allowedApproversIds | Read | List (Strings) | No |
| passwordPolicy | Read/Write | String | No |
| accountPolicy | Read/Write | String | No |
| respolResPwsdPolicy | Read/Write | String | No |

| Attribute | Editable? | Data Type | Required? |
|---|---|---|---|
| respolExcludeAccountsRule | Read/Write | String | No |
| retryMax | Read/Write | Integer | No |
| retryDelay | Read/Write | Integer | No |
| retryEmail | Read/Write | String | No |
| retryEmailThreshold | Read/Write | Integer | No |
| form | Read | String | No |
| licensedProducts | Read | List (Strings) | No |
| available.MetaViewAttribute | Read | List (Strings) | No |
| available.extendedAttributes | Read | List (Strings) | No |
| available.formFieldNames | Read | List (Strings) | No |

The resource viewer instantiates the resource parameters for the various view methods as follows:

- The `createView` method requires a typeString option, which is used to locate the correct prototypeXML for the resource type. The prototypeXML contains the initial set of resource parameters and their initial values. Thus, the view is populated with this list of initial resource parameters and their default values.
- The `getView` and `checkoutView` methods return only the resource parameters that exist in the resource object. The prototypeXML is not used to fill in this list if any resource parameters are missing in the actual resource object.
- The `checkinView` method replaces the list of resource parameters in the stored resource object in the repository. Again, the prototypeXML is not used to fill in any missing resource parameters that are not supplied during the `checkinView` operation.

## Resource Parameters

Resource parameters vary depending on the type of resource adapter being configured. Each resource contains a prototypeXML string that the resource viewer uses to determine the default set of resource parameters and their default values. Once Identity Manager creates a resource object, the resource viewer no longer uses the prototypeXML string, but rather uses the resource parameters from the actual object.

The following attributes uniquely identify the resource object.

### name

Externally identifies the resource. This user-supplied name is unique among resource objects.

### adapterClassName

Identifies the Resource Adapter class to be used to provision to the resource.

### type

Identifies the data type of the attribute.

### typeString

Specifies the internal name for the resource type.

### typeDisplayString

Identifies the display name for the resource type. This should be a message key or ID to be found in the message catalog.

### syncSource

If set to true, indicates that the resource supports synchronization events.

### facets

### description

Provides a textual description of the resource.

### startupType

Specifies whether the activeSync resource starts up automatically or manually.

Additional attributes depend upon the type of adapter being configured. At a minimum, these attributes specify how to connect to the resource. Typical parameters include TCP port, user, and password.

### host

Uniquely identifies the host.

### password

Specifies the password of the user (host administrator) to connect as.

### TCPPort

Identifies the port on the host to connect to.

### user

Identifies the user (host Administrator) to connect as.

## Account Attributes

These attributes define the accounts managed on this resource. Attributes vary depending on the resource type.

Typical attributes are:

### accountId

Specifies the ID by which the resource identifies this account.

### roles

Identifies the roles the account will have on the resource.

## Identity Template

The identity template is used to generate a user's identity on this resource.

## Identity Manager Parameters

Identity Manager parameters are used by Identity Manager to help manage the resource.

### resourceName

Specifies the name by which Identity Manager identifies this resource object.

### displayName

Specifies the display name that will display on the Identity Manager user edit and password pages to help identify users.

### retryMax

Indicates the maximum number of retries that will be tried on errors attempting to manage objects on a resource.

### retryDelay

Specifies the number of seconds between retries.

### retryEmail

Identifies the email addresses to send notifications to after reaching the retry notification threshold.

### retryEmailThreshold

Specifies the number of retries after which an email is sent.

### form

Identifies the user form that is used in workflows that edit accounts on the resource.

### passwordPolicy

Specifies the password policy for accounts on this resource.

### resourcePasswordPolicy

Indicates the resource password policy for resource accounts on this resource.

### accountPolicy

Specifies the policy for account IDs on this resource.

### excludedResourceAccountsPolicy

Specifies the policy for excluding resource accounts from account lists.

### approvers

Lists the administrator approvers for this resource.

### organizations

Lists the organizations available to the resource.

# Resource Object View

Used when modifying resource objects.

All attributes are editable, except `<resourceobjectType>.oldAttributes`, which are used to calculate attribute-level changes for updates.

In practice, replace `<resourceobjectType>` with the lowercase name of a resource-specific object type (for example, `group`, `organizationalunit`, `organization`, or `role`).

| Attribute | Editable? | Data Type | Required? |
|---|---|---|---|
| `resourceType` | Read/Write | String | |
| `resourceName` | Read/Write | String | |
| `resourceId` | Read/Write | String | |
| `objectType` | Read/Write | String | |
| `objectName` | Read/Write | String | |
| `objectId` | Read/Write | String | |
| `requestor` | Read/Write | String | |
| `attributes` | Read/Write | Object | |
| `oldAttributes` | Read | Object | |
| `organization` | Read/Write | String | |
| `attrstoget` | Read/Write | List | |
| `searchContext` | Read/Write | Object | |
| `searchAttributes` | Read/Write | List | |
| `searchTimelimit` | Read/Write | String | |

Table 31. ResourceObject View Attributes

## <resourceobjectType>.resourceType

Lists the Identity Manager resource type name (for example, LDAP, Active Directory).

## <resourceobjectType>.resourceName

Lists the Identity Manager resource name.

## <resourceobjectType>.resourceId

Lists the Identity Manager resource ID or name.

## <resourceobjectType>.objectType

Indicates the resource-specific object type (for example, `Group`).

## <resourceobjectType>.objectName

Lists the name of the resource object.

## <resourceobjectType>.objectId

Specifies the fully qualified name of the resource object (for example, `dn`).

## <resourceobjectType>.requestor

Specifies the ID of the user who is requesting the view.

## <resourceobjectType>.attributes

Indicates new or updated resource object attribute name/value pairs (object). This attribute has the following subattribute:

`resourceattrname --` String used to get or set the value of a specified resource attribute (for example, <***objectType***>`.attributes.cn`, where `cn` is the resource attribute `common name`).

## <resourceobjectType>.oldAttributes

Specifies the fetched resource object attribute name/value pairs (object). You cannot edit this value. The view uses this attribute to calculate attribute-level changes for update.

## <resourceobjectType>.organization

Identifies the list of organizations of which the resource is a member. This list is used to determine which organizations should have access to the associated audit event record when available for future analysis and reporting.

## <resourceobjectType>.attrstoget

List of object-type-specific attributes to return when requesting an object with the `checkoutView` or `getView` methods.

## <resourceobjectType>.searchContext

Specifies the context used to search for non-fully qualified names in resources with hierarchical namespaces.

## <resourceobjectType>.searchAttributes

Lists the resource object type-specific attribute names that will be used to search within the specified `searchContext` for names of resources with hierarchical namespaces.

## <resourceobjectType>.searchTimelimit

Specifies the maximum time spent searching for a name input to a form (if supported by the resource).

# Role View

Used to define Identity Manager role objects.

When checked in, this view launches the Manage Role workflow. By default, this workflow simply commits the view changes to the repository, but it also provides hooks for approvals and other customizations.

The following table lists the high-level attributes of this view.

| Attribute | Editable? | Data Type | Required |
|---|---|---|---|
| name | Read/Write | String | Yes |
| resources | Read/Write | List | No |
| applications | Read/Write | List | No |
| roles | Read/Write | List | No |
| assignedResources | Read/Write | List | No |
| notifications | Read/Write | List | No |
| approvers | Read/Write | List | No |
| properties | Read/Write | List | |
| organizations | Read/Write | List | Yes |

Table 32. Role View Attributes

## name

Identifies the name of the role. This corresponds to the name of a Role object in the Identity Manager repository.

## resources

Specifies the names of locally assigned resources.

## applications

Specifies the names of locally assigned applications (Resource Groups).

## roles

Specifies the names of locally assigned roles.

## assignedResources

Flattened list of all assigned resources via resources, applications, and roles.

| Attribute | Editable? | Data Type |
|---|---|---|
| resourceName | | String |
| attributes | | Object |

### resource name

Identifies the name of the assigned resource.

### attributes

Identifies the characteristics of the resource. All subattributes are strings and are editable.

| Attribute | Description |
|---|---|
| name | Name of resource attribute |
| valueType | Type of value set for this attribute. Allowed values include Rule, text, or none. |
| requirement | Type of value set by this attribute. allowed values include Rule, Text, None, Value, Merge with Value, Remove with Value, Merge with Value clear existing, Authoritative set to value. Authoritative merge with value, Authoritative merge with value clear existing. |
| rule | Specifies rule name if value type is Rule. |
| value | Specifies value if rule type is Text. |

Table 33. attribute Options (Role View)

## notifications

Lists the names of administrators that must approve the assignment of this role to a user.

## approvers

Specifies the names of the approvers that must approve the assignment of this role to a user.

## properties

Identifies the user-defined properties that are stored on this role.

## organizations

Lists organizations of which this role is a member.

# Task Schedule View

Use to create and modify TaskSchedule objects.

This view contains the following attributes:

| Name | Editable? | Data Type | Required? |
|---|---|---|---|
| scheduler | Read/Write | String | |
| task | Read/Write | Boolean | |

Figure 25. Task Schedule View Attributes

## scheduler

Contains attributes that are related to the scheduler itself, which are common to all scheduled tasks. The attributes are:

| Name | Editable? | Data Type | Required? |
|---|---|---|---|
| name | Read/Write | String | No |
| id | Read | String | No |
| definition | Read/Write | String | No |
| template | Read/Write | String | No |
| taskOrganization | Read/Write | String | No |
| taskName | Read/Write | String | No |
| description | Read/Write | String | No |
| disabled | Read/Write | Boolean | No |
| skipMissed | Read/Write | Boolean | No |
| start | Read/Write | Date | No |
| repeatCount | Read/Write | Int | No |
| repeatUnit | Read/Write | String | No |
| resultOption | Read/Write | String | No |
| allowMultiple | Read/Write | Boolean | No |

Figure 26. scheduler Attributes (Task Schedule View)

**Note** Typically, you supply a value for either `scheduler.definition` or `scheduler.template`. If you do not specify either value, Identity Manager creates a TaskSchedule object that you can later edit to specify the definition or template.

## name

Specifies the name of an existing TaskSchedule object or the desired name for a new TaskSchedule object. It is not required, but if not specified, the system will generate a random identifier.

## id

Uniquely identifies the existing TaskSchedule object.

## definition

Defines the name a TaskDefinition object to be scheduled.

## template

Specifies the name of a TaskTemplate object to be scheduled. If both `definition` and `template` are specified, `template` has priority.

## taskOrganization

Contains the name of the organization in which the TaskInstance will be placed when the schedule task is launched.

## taskName

Specifies the name of the TaskInstance that is created when the schedule task is launched.

## description

Contains descriptive text that will be saved in the TaskInstance that will be created when the schedule task is launched. The description will appear in the task tables in the product interface.

## disabled

Controls whether the task scheduler will process the TaskSchedule object. The scheduler ignores TaskSchedule's whose disable attribute is true. You can use this to temporarily stop running a schedule task, without having to delete and recreate the TaskSchedule object.

## start

Indicates the date and time at which to launch the task.

## repeatCount

Combined with `repeatUnit`, determines how frequently tasks will be run. If repeatCount is zero or not specified a scheduled task will only run once. If repeatCount is a positive number, the task will be run more than once at the interval specified by repeatUnit.

## repeatUnit

Defines the interval of time between running tasks that have a positive `repeatCount` value. Valid values include: `second`, `minute`, `hour`, `day`, `week`, `month`. For example, to schedule a task to run once a week for a year set `repeatUnit` to week, `repeatCount` to 52, and `start` to the first day that the task is to run.

## resultOption

Specifies what the scheduler will do if a TaskInstance with the desired name already exists when the scheduled task is run. The possible values are: `wait`, `delete`, `rename`, and `terminate`.

## wait

Indicates whether the scheduler should run the task again or wait for another repetition. This attribute is only meaningful if you have set `repeatCount` and `repeatUnit`.

## delete

Tells the scheduler to delete the existing TaskInstance, if it has finished.

## rename

Indicates that the scheduler should rename the existing TaskInstance, if it has finished.

## skipMissed

Indicates whether Identity Manager attempts to immediately make up a missed schedule time (`false`) or simply wait until the next scheduled time (`true`).

When set to `false`, Identity Manager immediately attempts to make up a missed schedule time. When set to true, Identity Manager instead waits until the next scheduled time. The default is `false`.

### terminate

Similar to delete, but will also terminate the existing task, if it is still running.

### allowMultiple

Controls whether more than one instance of the same task definition or task template are allowed to run. If `true` (the default), the scheduler will always create a new instance of the task. If `false`, the scheduler will not create a new instance if there is one already running.

## task

Contains task-specific attributes. Each task defines its own attributes, and the task's form should reference them relative to the `task` namespace.

# Unlock View

Used to unlock accounts for those resources that support native account locking. This view presents and selects the list of resource accounts to be unlocked.

**Note** Use the Unlock view instead of the Disable view for accounts whose resources support native account locking.

Contains the following high-level attributes:

| Name | Editable? | Data Type | Required? |
|---|---|---|---|
| id | Read | String | Yes |
| selectAll | Read/Write | Boolean | No |
| currentResourceAccounts | Read | List (objects) | No |
| tobeCreatedResourceAccounts | Read | List (objects) | No |
| tobeDeletedResourceAccounts | Read | List (objects) | No |

Figure 27. Unlock View Attributes

## id

Specifies the account ID of the Identity Manager user whose passwords are being unlocked.

## selectAll

Controls whether all password are unlocked.

## currentResourceAccounts

Represents the set of accounts that are currently being managed by Identity Manager (including the Identity Manager account itself).

## tobeCreatedResourceAccounts

Represents the accounts that are assigned to this Identity Manager user but which have not been created. Passwords cannot be unlocked on accounts that have not yet been created.

## tobeDeletedResourceAccounts

Represents the accounts that have been created but are no longer assigned to this user. Passwords cannot be changed on accounts that are going to be deleted.

All three account lists contain objects that describe the state of the account on each resource and allow you to individually select accounts.

Both resource account list are indexed by resource name, and will contain objects that describe the resources on which this user has accounts.

| Name | Editable? | Data Type |
|---|---|---|
| `selected` | Read/Write | Boolean |
| `name` | Read/Write | String |
| `type` | Read/Write | String |
| `accountId` | Read/Write | String |
| `exists` | Read/Write | Boolean |
| `locked` | Read/Write | Boolean |
| `authenticator` | Read/Write | Boolean |

Figure 28. tobeDeletedResourceAccounts Attributes (Unlock View)

### selected

Identifies that this resource has been selected to be unlocked.

### name

Specifies the name of resource. This corresponds to the name of a `resource` object in the Identity Manager repository

### type

Identifies the type of resource, such as Solaris. You can determine the resource type names by bringing up the resource list from the Identity Manager Administrator interface. The **Type** column on this page contains the names of the type of currently defined resources. The options list next to **New Resource** also contains the names of the resource adapters that are currently installed.

### accountId

Specifies the identity of the account on this resource, if one has been created.

### exists

Indicates whether the account already exists on the resource (only in `currentResourceAccounts`).

### locked

Indicates whether the account is currently locked or not (unlocked). The value of `exists` indicates whether the account already exists on the resource or not (only in `currentResourceAccounts`).

### authenticator

If `true`, indicates that this resource serves as the pass-through authentication resource for Identity Manager.

# WorkItem View

Used to view and modify WorkItem objects in the repository.

A *WorkItem* object is created whenever a manual action that is defined in a workflow process is activated. The WorkItem view contains a few attributes that describe the WorkItem object itself, as well as values of selected workflow variables copied from the workflow task.

Identity Manager returns information about the work items in the Work Item view under the `workItem.related` attribute.

## Returning Information about All Active Work Items

This view provides the ability to return information about all work items that are currently active in a workflow task. By default, Identity Manager returns information about only a specified work item, not related work items. However, you can use other options to filter work items, and the attributes of the related work items you want to display.

Use the following three form properties to change the default behavior of this view:

| If you want to ... | Use this form property |
|---|---|
| Return all related items by default... | `includeRelatedItems` form property |
| Request additional attributes to be returned... | `relatedItemAttributes` form property |
| Limit which items are returned... | `relatedItemFilter` form property |

### Example: Using the includeRelatedItems Form Property

By default, Identity Manager uses the Approval form to display work items. Edit this form by adding the `includeRelatedItems` element to include related work items:

```
<Properties>
   <Property name='includeRelatedItems' value='true'/>
</Properties>
```

## Example: Using the relatedItemAttributes Form Property

You can also request additional attributes with the `relatedItemAttributes` option. This option can be a CSV string of names or a list of names. You can request the following standard attributes:

- `request`
- `requester`
- `description`
- `activityName`

If you request an attribute name that is not on this list, Identity Manager assumes that it is an arbitrary workflow variable, and the value will be returned if it exists in the work item. Common variables found in the standard workflows include:

- `accountId`
- `objectType`
- `objectName`
- `diagramLabel`

## Example: Using the includeRelatedItems Form Property

To include the `request` and `description` attributes, add these properties to the Approval form:

```
<Properties>
   <Property name='includeRelatedItems' value='true'/>
   <Property name='relatedItemAttributes'
value='request,description'/>
</Properties>
```

## Example: Using relatedItemFilter Form Property

You can specify the following filter attributes.

| relatedItemFilter Option Values | Results of Filtering |
|---|---|
| `itemType` | Only work items with a matching itemType are returned |
| `activityName` | Only work items created from the same activity are returned |
| `request` | Only work items with the same user defined request string are returned |
| `locked` | Only work items that are currently locked for editing are returned |

If more than one filter attribute is on the list, they will be logically AND'd together. For example, to return only work items with the same request string that are current locked, add this property to the Approval form:

```
<Properties>
   <Property name='includeRelatedItems' value='true'/>
   <Property name='relatedItemAttributes'value='request,description'/>
   <Property name='relatedItemFilter' value='request,locked'/>
</Properties>
```

An example field that displays a table of information about the related work items was added to the Approval Library form library, the field name is Related Approvers. You can reference this field from the standard Approval form as follows:

```
 <FieldRef name='Related Approvers'/>
```

## Changing the Repository Lock Timeout for Work Items

The default time-out interval for locking work items in the repository is five minutes. You can change this value by adding the following element to the `RelocatedTypes` element of the RepositoryConfiguration Configuration object:

```
<TypeDataStore typeName='WorkItem' lockTimeoutMillis='10000'/>
```

## Top-Level Attributes

The following table lists the top-level WorkItem view attributes.

| Attribute | Editable? | Data Type | Required? |
|---|---|---|---|
| id | Read | String | |
| name | Read | String | |
| taskId | Read | String | |
| taskName | Read | String | |
| processName | Read | String | |
| activityName | Read | String | |
| description | Read/Write | String | |
| owner | Read/Write | String | |
| complete | Read/Write | Boolean | |
| variables | | | |
| workItem | | | |

Table 34. WorkItem View Attributes

## id

Identifies the repository ID of the WorkItem object. Typically generated by Identity Manager and not displayed.

## name

Identifies the repository name of the WorkItem object.

## taskId

Identifies the repository ID of the workflow TaskInstance. This attribute is used by the system to correlate the work item with the workflow task and must not be changed.

## taskName

Identifies the repository name of the workflow TaskInstance. This name is typically set to an informative value and can be displayed. Do not modify it. A typical example task name for a user update would be `Updating User jdoe`.

## processName

Identifies the name of the workflow process definition that contains the manual action.

## activityName

Specifies the name of the workflow activity that contains the manual action.

## description

Contains a textual description of the work item. Its contents are defined by the workflow process definition. The description is typically displayed in tables that summarize the work items for a user, and is often displayed in a work item form.

## owner

Identifies the name of the current Identity Manager administrator or user that created the workflow process. This attribute is typically the name of an Identity Manager user. If this work item is assigned to an anonymous user, the name will have the prefix **Temp:**.

## complete

Set to `true` when the manual action has completed and the workflow is to be resumed. Assignment of the complete attribute must be performed in the Work Item form.

You can edit this Boolean value.

## variables

Contains another object whose attributes contain copies of variables from the workflow task. By default, every workflow variable that is in scope when the manual action is activated is copied into the work item. This can be controlled with the Exposed Variables and Editable Variables options in the process definition. Most work item forms display information found under the `variables` attribute. See the section *Using the variables Attribute* later in this chapter for more information on using this attribute.

## workItem

Specifies additional information about the work item. Contains the following attributes:

### views

Contains a list of workflow variables whose values are views. The system uses this attribute to cause view-specific refresh operations when the work item view is refreshed.

Do not change this value.

### related

Contains a list of attributes that describe the specified work item.

| Attribute | Description |
|---|---|
| `name` | Specifies the repository ID of the work item. |
| `owner` | Identifies the owner of the item. |
| `locked` | Indicates whether the work item is being edited. A value of `true` indicates that the work item is currently being edited. |
| `complete` | Specifies whether the work item has completed. A value of `true` indicates that the work item completed. |
| `itemType` | Identifies item type as defined by the process. The default is `approval`. |

Table 35. Subattributes of the workItem.related Attribute (Work Item View)

### request

Succinctly describes the purpose of the work item. This description is typically shorter than the value of the `description` attribute and is often displayed in summary tables.

### requester

Identifies the user that initiated the approval.

### ignoreTimeOut

Indicates whether the time out should be ignored. A value of `true` (assigned by the system) indicates that this is a read-only work item that may timeout while being viewed. This is a signal to the system that a check-in failure of the Work Item view should be ignored if the work item no longer exists, rather than displaying an error

message. This can be useful for work items that are intended only for status messages that time out immediately so the workflow can continue while the user views the messages.

Do not change this value.

## Using the variables Attribute

When writing a work item form, the most common attributes to reference are `complete` and `variables`. The complete attribute must be set to the value `true` in order for the workflow to be resumed. It is typically set by a hidden field in response to pressing button fields with labels such as **Approve** and **Reject**.

The `variables` attribute contains an object whose values are copies of variables from the workflow task. One of the most common workflow variables used in work items is `user`, which contains a user view. For example, to reference the `global.email` attribute from a work item form, use the following path expression:

```
variables.user.global.email
```

This differs from attribute paths used in a standard user form. First, the entire view is stored in a workflow variable named `user`, which results in the `user.` prefix being required in the attribute path. Next, the workflow variables are stored under the `variables` attribute in the Work Item view, which results in an additional `variables.` prefix being required in the attribute path.

Because of this nesting of the user view attributes, you cannot use a standard user form with the Work Item view without modification. However, you can define a work item form that references the user form with the `base context` option.

## Example

```
<Form name='WorkItemForm'>
   <Include>
      <ObjectRef Type='UserForm' name='Default User Form'/>
   </Include>
   <FormRef name='Default User Form' baseContext='variables.user'/>
</Form>
```

**Note** Although in practice the work item form requires additional fields for buttons such as Approve and Reject, you may not want everything displayed by Default User Form displayed in the work item form. Typically, you can factor out the fields in the user form into a form library that can be referenced by both the user forms and the work item forms.

# WorkItem List View

Used to view information about collections of work items in the repository and to perform operations on multiple work items at a time.

This view handler gathers information about:

- all work items assigned to a selected user
- users whose work items can be viewed
- users to whom the work items can be forwarded

The view is used in the Approvals page of the Identity Manager Administrator Interface. The default form used with this view is named Work Item List.

The following table lists the top-level WorkItem List view attributes.

| Attribute | Editable? | Data Type |
|---|---|---|
| authType | Read/Write | String |
| userId | Read | String |
| user | Read/Write | String |
| self | Read | Boolean |
| forwardedUser | Read | Boolean |
| itemType | Read/Write | String |
| users | Read | List |
| userIds | Read | String |
| forwardingApproverStyle | Read | |
| forwardingUsers | Read | List |
| forwardingUserIds | Read | List |
| workItems | Read/Write | String |
| selectedWorkItems | Read/Write | String |
| forwardTo | Read/Write | Boolean |
| forwardToNow | Read/Write | String |
| variables | Read/Write | String |
| action | Read/Write | Boolean |
| confirm | Read/Write | Boolean |

Table 36. WorkItem View Attributes

## authType

Specifies access to work items by type. For example, there is a built-in authorization type called EndUserRule. All end-users implicitly get access to all rules tagged with the EndUserRule authorization type.

## userId

Specifies the name of the Identity Manager user whose work items are contained in the workItem list. Initially, this value is the name of the current session user. The value can be null to indicate that the work items for all controlled users with approver rights should be displayed. This is always the Identity Manager user name, never a display name.

The form must not be modify this value. To change users, set the `user` attribute.

## user

Specifies the display name of the Identity Manager user whose work items are listed. This value is the same as `userId` if display names are not used. The form can modify this value, which causes the system to recalculate the work item list during refresh. A null value indicates that all work items are being displayed.

## self

Set to true if the `userId` is the same as the current session user.

## forwardedUser

When set, indicates that the user named by `userId` has elected to have work items forwarded to another user. The other user is identified by its display name.

## users

Lists the display names of Identity Manager users that the current user controls and which have work item capabilities. This value is typically used to build an user select box. If a custom form wants to compute the user list in a different way, you can specify the view option CustomUserLists as either a view option or form property.

## userIds

Typically null. If you are configured to use alternate display names, then the users list contains display names, and this list contains the true repository names.

## forwardingUsers

Lists the display names of Identity Manager users to which the current user can forward work items. This is defined as similar users who have both WorkItem rights and control at least one of the same organizations as the current user.

## itemType

When set, the work items in the list will be filtered to contain only those whose item type matches this value. This gives the WorkItemList view the ability to filter the item list based on the work item type.

## forwardingUserIds

Typically null. If you are configured to use alternate display names, then the `forwardingUsers` list will have display names, and this list will have the true repository names.

## workItems

Lists the objects that contain information about the work items for the selected user(s). The object names are the repository IDs of the work items.

### workItems[].owner

Specifies the display name of the owner. Set only if `user` is null and all work items are displayed.

### workItems[].request

Supplies a brief description of the object being requested. This value is computed by the WorkItemRequest expression of the manual action in the workflow process.

### workItems[].requester

Identifies the display name of the user that made the request.

### workItems[].description

Provides a more detailed description of the work item. The value is computed by the WorkItemDescription expression of the manual action in the workflow process. The description is typically displayed in tables that summarize the work items for a user, and is often displayed in a work item form.

### workItems[].selected

Individual item selection flag. An alternative to `selectedWorkItems`.

## selectedWorkItems

Lists the work item IDs that represent the items to be processed by the next action. An alternative to setting the `selected` attribute inside the work item object, which is easier for SortingTable components. If both this attribute and individual select flags are set, the value of this attribute takes precedence.

## forwardTo

Identifies the name of an Identity Manager user to which all selected work items will be forwarded when the `action` attribute is set to `Forward`.

## forwardToNow

Similar to `forwardTo`, but is also an `action` attribute. It copies its value to `forwardTo`, set `action=Forward` and process the refresh as if `forwardTo` and `action` were set independently. Use this attribute if you want to have the form process the forwarding immediately after a user is selected from a form component. If you would rather have forwarding controlled with a button, then have the form component set the `forwardTo` attribute and have the button post an `action` value of `Forward`.

## action

(Boolean) When non-null, initiates an operation on the selected work items.

Valid values include:

- approve
- reject
- forward
- refresh

If the NoConfirm option is set, the action is processed immediately. Otherwise, Identity Manager waits for the `confirm` attribute to be set to `true`. The form is expected to define its own confirmation page rendering.

### confirm

(Boolean) Indicates that the operation specified in the `action` attribute can be performed.

### Using the variables Attribute

When editing an individual work item, the form can set work item variables, such as `comments`, to pass additional information about the approval or rejection into the workflow process for auditing.

You can also set arbitrary work item variables when performing actions in the WorkItemList view. The value of the attribute `variables` can be set to an object whose attributes will be copied into the work item when it is approved or rejected. For example, if the variables object contains an attribute named `comments`, the same comments will be saved with every selected work item.

#### Example

```
<Form name='variables.comments'>
  <Default>
    <concat>
      <s>Approval performed on </s>
      <invoke class='com.waveset.util.Util' name='dateToString'>
        <new class='java.util.Date'/>
      </invoke>
    </concat>
  </Default>
</Form>
```

**Note** Although in practice the work item form requires additional fields for buttons such as Approve and Reject, you may not want everything displayed by Default User Form displayed in the work item form. Typically, you can factor out the fields in the user form into a form library that can be referenced by both the user forms and the work item forms.

## View Options

You can specify the following options when the view is created or refreshed to control the behavior of the WorkItemList viewer.

### userId

Identifies the name of the initial user whose work items are to be displayed. Can be used to override the default, which is the current session user.

## CustomUserLists

When set to `true`, indicates the form will generate both the `users` and `forwardingUsers` lists in a custom way and that the view handler should not generate them. Generating these lists can be time-consuming if there are many approvers in the system. If the form does not intend to use the default `users` and `forwardingUsers` lists, enable this option.

## ForwardingApproverStyle

Specifies the types of administrators whose names will be available in the Forward to list. Can be set to one of these values:

| Option Value | Description |
|---|---|
| `peers` | Specifies administrators at the same organization level as the current user or above |
| `controlled` | Specifies administrators in organizations that are controlled by the current user |
| `all` | Specifies both controlled and peers |

You can set this and other view options as form properties:

```
<Form...>
   <Properties>
      <Property name='ForwardingApproverStyle" value='peers'/>
   </Properties>
     ...
</Form>
```

## NoUserListCache

When `true`, indicates that the view handler should not cache the `users` and `forwardingUsers` lists but instead recalculate them every time the form is refreshed. Since calculating the user lists can be expensive, it is generally preferred to cache them and refresh only when explicitly instructed by setting the `action` attribute to Refresh.

### UserDisplayName

Can be set to the name of an extended user attribute whose value is to be used instead of the repository name in the user lists. This can also be specified in the UserUIConfig object, but it may be more convenient to set in the form.

### NoUserDisplayName

When `true`, indicates that display names should not be used even if one is specified in the UserUIConfig object. You can set this option in a form to selectively override the UserUIConfig setting.

### NoConfirm

When `true`, indicates that the action specified with the `action` attribute should be executed immediately without confirmation.

## Setting View Options in Forms

View options can be conveniently set in the form used to render the Approvals page. To customize this form

1. Copy the form named Work Item List into your XML editor of choice.
2. Change the name.
3. Register it in the System Configuration object under the `form.workItemList` attribute.

In the custom form, you can then specify view options as properties of the form.

### Example

```
<Form>
    <Properties>
      <Property name='CustomUserLists' value='true'/>
    </Properties>

     ...
  </Form>
```

# Deferred Attributes

A *deferred attribute* is an attribute that derives its value from an attribute value on a different account. You declare the deferred attribute in a view (and the WSUser model), and the provisioning engine performs this substitution immediately before calling the adapter.

If the deferred attribute derives its value from another resource's GUID attribute, the source adapter does not need to take action. However, if the source attribute is not the GUID, the adapter must return the attribute in the ResourceInfo._resultsAttributes map as a side effect of the realCreate operation. If the adapter does not return the attribute, the provisioning engine will fetch the account to get the value. This is less efficient than modifying the adapter to return the value.

## When to Use Deferred Attributes

Use deferred attributes when creating new accounts to specify that the value of an account attribute is to be derived from the value of an attribute on a different account that will not be known until the source account has been created. One common example is to set an attribute to the value of the generated unique identifier.

## Using Deferred Attributes

There are two main steps to defining a deferred attribute:

1. Ensure that the account is created on the source resource before the second account is created. Do this by creating an ordered Resource Group that contains both resources and assigning the Resource Group to the user.
2. Set the special attributes in the User view for the accounts that are to be created as indicated by the following sample scenario. Each deferred attribute requires two view attributes: one that identifies the source account, and one that identifies the source attribute. Set these using paths of the following form:

   ```
   accounts[<resource>].deferredAttributes.<attname>.resource
   ```

```
accounts[<resource>].deferredAttributes.<attname>.attribute
```

where `<resource>` would be replaced with an actual resource name and `<attname>` replaced with an actual attribute name.

For example, assume a scenario in which the following two resources are created: 1) a resource named LDAP that generates a `uid` attribute when an account is created; 2)a resource named HR, which contains a `directoryid` attribute named `directoryid`, whose value is to be the same as `uid` in the LDAP resource.

The following form fields set the necessary view attributes to define this association.

```
<Field
name='accounts[HR].deferredAttributes.directoryid.resource'>
   <Expansion><s>LDAP</s></Expansion>
</Field>
<Field name='accounts[HR].deferredAttributes.directoryid
   <Expansion><s>uid</s></Expansion>
</Field>
```

# Extending Views

Some views that set specific resource account attributes such as the password or the enable flag allow you to set additional account attributes. For security, however, these extended attributes must be registered.

## Attribute Registration

Attributes can be registered in one of two locations:

| Location | Register attributes here if... |
|---|---|
| `AccountAttributeType` definition in the resource | ... the attributes you want to update are specific to a particular resource, rather than to all resources of that type. |
| System Configuration Object | ...you want to make global registrations for all resources of a particular type. These registrations must be done in XML format. |

Table 37. Locations for Attribute Registration

You can register different attributes for different views. For example, you can register the `lock` attribute for the Password view and the `firstname` attribute for the Rename view.

### Global Registration

To make global registrations (that is, registrations that apply to all resources), add an attribute in the System Configuration object with this path:

```
updatableAttributes.ViewName.ResourceTypeName
```

where ***ViewName*** is one of Password, Reset, Enable, Disable, Rename, or Delete, and ***ResourceTypeName*** is the name of the resource type. The type name `all` is reserved for registrations that apply to all resources.

The value of this attribute must be a List of Strings. The strings are names of the attributes you want to update.

The following example registers the attribute named `delete before action` in the Deprovision view for all resources.

```
<Attribute name='updatableAttributes'>
   <Object>
      <Attribute name='Delete'>
         <Object>
            <Attribute name='all'>
               <List>
                  <String>delete before action</String>
               </List>
            </Attribute>
         </Object>
      </Attribute>
      <Attribute name='Enable'>
         <Object>
            <Attribute name='all'>
               <List>
                  <String>enable before action</String>
               </List>
            </Attribute>
         </Object>
      </Attribute>
   </Object>
</Attribute>
```

## Resource-Specific Registration

To make resource-specific registrations, modify the resource object from the Identity Manager Debug page and insert a `<Views>` subelement in the `AccountAttributeType` element. `<Views>` must contain a list of strings whose values are the names of the views in which this attribute can be updated.

```
<AccountAttributeType name='lastname' mapName='sn'mapType='string'>
   <Views>
      <String>Rename</String>
   </Views>
</AccountAttributeType>
```

In the view, attributes you want to modify are placed within this object:

`resourceAccounts.currentResourceAccounts[`***ResourceTypeName***`].attr ibutes`

## Example

```
<Field name=
'resourceAccounts.currentResourceAccounts[OS400ResourceName].attribut
es.delete before action' hidden='true'>
   <Expansion>
      <s>os400BeforeDeleteAction</s>
   </Expansion>
</Field>
```

# 6 XPRESS Language

This chapter introduces the basic features of XPRESS, an XML-based expression and scripting language. Statements written in this language, called *expressions*, are used throughout Identity Manager to add data transformation capabilities to forms and to incorporate state transition logic within objects such as workflow and forms.

The chapter covers the following basic topics related to writing expressions with the XPRESS language:

- **Overview** – Describes some essential features of the XPRESS language, including its use of prefix notation and XML syntax.
- **Working with Expressions** – Presents examples of the more common uses of expressions within Identity Manager.
- **Functions** – Lists the library of functions that ships with Identity Manager.
- **Data Types** – Identifies the possible data types that functions return.

## Overview

XPRESS is a *functional* language, similar in structure to Lisp, that uses syntax based on XML. Every statement in the language is a function call that takes zero or more arguments and returns a value. A rich set of built-in functions is provided, and you can also define new functions. XPRESS also supports the invocation of methods on any Java class and the evaluation of JavaScript within an expression.

This introductory section covers the following topics related to the XPRESS language:

- Prefix notation
- XML syntax
- Integration with Identity Manager application

# Prefix Notation

Like Lisp, the XPRESS language makes no distinction between a *function call* and what languages such as C refer to as an *expression operator*. This results in a syntactical style known as *prefix notation*. Prefix notation differs from the more common *infix* notation in that the operator of an expression is written first, followed by the operands. For example, consider the following simple logical expression written in C using infix notation:

```
x == 42
```

If C used prefix notation, the previous statement would be written:

```
== x 42
```

If C provided no expression operators and instead supplied only functions, the statement could be written as follows:

```
equals(x, 42)
```

Prefix notation is easily understood if you think in terms of calling functions rather than writing expressions.

# XML Syntax and Example

XPRESS uses an XML syntax that is easy to parse and manipulate and can be embedded naturally in other XML vocabularies used within Identity Manager. The names of the XML elements are the names of functions to be called. Nested elements are the arguments to the function. In addition, there are beginning and end tags for each element (in this case, `<add></add>`).

**Example**

```
<add> <ref>counter</ref> <i>10</i> </add>
```

In the preceding example, the `<add>` element represents a call to the function named `add`. This function is passed two arguments:

- *first argument* – value is determined by calling a function named `ref`. The argument to the `ref` function is a literal string that is assumed to be the name of a variable. The value returned by the `ref` function is the current value of the variable *counter*.
- *second argument* – value is determined by calling a function named `i`. The argument to the `i` function is a literal string that is an integer. The value that the `i` function returns is the integer 10.

The value returned by the `add` function will then be the result of adding the integer 10 to the current value of the variable *counter*. Every function call either returns a value or performs an operation on one of its arguments. For example, if the `ref` call returns the value of the counter, then the `<i>` call returns the integer `10`, and the `<add>` call returns the addition of the two calls.

Another example is the classic `Hello World` program, which is written in XPRESS as follows:

```
<print><s>Hello World!</s></print>
```

## Integration with Identity Manager

Although XPRESS can be used with a standalone interpreter, it is typically embedded within an application that wants to use XPRESS statements to control or customize their behavior. This application is called the *host application*. Two of the more important host applications within the Identity Manager system are workflow and forms.

The host application makes calls to the XPRESS interpreter and supplies services to the interpreter. One of the more important services that the host application provides is the resolution of *external variable references*. Expressions often reference variables that are not defined within the expression, and the host application must then provide the values of these variables. In the case of the workflow host application, an expression can reference any variable defined within the workflow process. In the forms host application, an expression can reference the value of any form field or `defvar` whose value is set before the expression is evaluated.

For information on testing XML syntax without checking the XPRESS logic of your statements, see the section titled Checking the XML Syntax of Expressions.

## Why Use Expressions?

Expressions are used primarily for the following tasks:

- **Customizing the end-user and administrator forms**. Forms use XPRESS to control the visibility of fields and to transform the data to be displayed.
- **Defining flow of control in workflow**. Workflow uses XPRESS to define *transition conditions,* which determine the order in which steps in the workflow process are performed.
- **Implementing workflow actions**. Workflow actions can be implemented using XPRESS. Action expressions can perform simple calculations, or call out to Java classes or JavaScript to perform a complex operation.

For information on using expressions in workflow scripts or editing forms, see *Identity Manager Technical Deployment*.

# Working with Expressions

This section presents examples of some of the more common usages of expressions within the Identity Manager system. It introduces examples of using expressions to perform the following:

- Controlling field visibility
- Calculating default field values
- Deriving field values
- Generating field values
- Workflow transition conditions
- Workflow actions
- Invoking Java methods from workflow actions

## Controlling Field Visibility

A common form design problem requires suppressing the display of certain fields until a particular condition is met. For example, certain resource-specific fields are relevant only when a particular resources is assigned to the user. These fields should be visible only when the resource is assigned. Otherwise, these fields should be hidden from view and not evaluated. The following example illustrates a field definition that uses an expression within the `<Disable>` element to control the visibility of such a field.

```
<Field name='HomeDirectory'>
   <Display class='Text'/>
      <Property name='title' value='HomeDirectory'/>
   </Display>
   <Disable>
      <not>
         <contains>
            <ref>accountInfo.typeNames</ref>
            <s>Solaris</s>
         </contains>
      </not>
   </Disable>
</Field>
```

The <Disable> element is part of the Form XML language. The contents of the <Disable> element can be any expression in the XPRESS language. In this case, the expression is testing to see if the string Solaris appears in a list stored in the external variable named accountInfo.typeNames. With forms, this variable contains a list of all resource types currently assigned to the user.

When the form is processed for display, the expression in the <Disable> element is evaluated. If it returns true, this field is not displayed.

The values null and 0 are logically false. Non-null or non-zero fields are logically true. This means that the sting represented with the expression <s>false</s> is logically true since it is non-null.

Field values can be calculated by XPRESS using one of three elements specified in the field declaration: Derivation, Default, and Expansion.

## Calculating Default Field Values

Field values can be calculated from other fields or simply set to an initial value using the <Default> element. The <Default> element is typically used to initialize an editable field and is evaluated only if the field does not already have a value assigned to it. The <Default> element is often used to calculate an account ID based on the first and last name of the user. The following example shows a field definition that uses string manipulation expressions to calculate a default account ID consisting of the first letter of the user's first name concatenated with the user's last name.

```
<Field name='waveset.accountId'>
   <Display class='Text'/>
      <Property name='title' value='AccountID'/>
   </Display>
   <Default>
      <concat>
         <substr>
            <ref>accounts[Exchange].firstname</ref>
            <i>0</i>
            <i>1</i>
         </substr>
         <ref>accounts[Exchange].lastname</ref>
      </concat>
   </Default>
</Field>
```

The <Default> element is part of the Form XML language. This element can contain either an XPRESS expression or elements in another language called *XML Object*. (For more information on XML Object language, see the chapter titled XML Object Language.)

When this field is processed, the system checks to see if a value already exists for the `waveset.accountId` attribute. If no value exists, it evaluates the expression in the `<Default>` element. In this case, a value is calculated by concatenating the first letter of the first name with the last name.

You may need to make sure that `firstname` and `lastname` fields have values, as demonstrated by the following example:

```
<cond>
  <and>
    <notnull><ref>accounts[Exchange].firstname</ref></notnull>
    <notnull><ref>accounts[Exchange].lastname</ref></notnull>
  </and>
  <concat>
    <substr>
         <ref>accounts[Exchange].firstname</ref>
         <i>0</i>
         <i>1</i>
     </substr>
    <ref>accounts[Exchange].lastname</ref>
  </concat>
</cond>
```

The preceding code is structured as an if-then statement in other programming languages. This `cond` expression has two arguments:

- conditional expression
- then expression

First, the conditional expression is evaluated. If the result of this expression is logically true, the value of `cond` will be the value of the then expression. If the result of the conditional expression is false, the value of `cond` will be `null`.

In this example, the `cond` statement ensures that values exist for two account attributes before using them to calculate `accountID`. The `Default` expression will continue to be evaluated each time the form is refreshed or saved until the prerequisites are finally set or until the user provides a value in the field. The `Default` expression will not be evaluated if the associated field contains a non-null value.

# Deriving Field Values

A `<Derivation>` expression is similar to a `<Default>` expression except that it always calculates a value for the field, even if the field already has a non-null value. This is typically used to display a field whose value is a permutation of another field's value. This is a valuable design feature if the resource attribute value is encoded and would not be obvious to the user.

The following example shows a field definition that uses conditional logic to map one set of values into another set.

```
<Field name='location' prompt='Location'>
   <Display class='Text'/>
      <Derivation>
         <switch>
            <ref>accounts[Oracle].locCode</ref>
            <case>
               <s>AUS</s>
               <s>Austin</s>
            </case>
            <case>
               <s>HOU</s>
               <s>Houston</s>
            </case>
            <case>
               <s>DAL</s>
               <s>Dallas</s>
            </case>
            <case default='true'>
               <s>unknown</s>
            </case>
         </switch>
   </Derivation>
</Field>
```

The `<Derivation>` element is part of the Form XML language that can contain an expression. When this field is processed, the expression in the `<Derivation>` element is evaluated to determine the value to be displayed for this field.

In the preceding example, the value of the resource account attribute `accounts[Oracle].locCode` is compared to the first value in each case expression. If a match is found, the result of the `switch` expression is the second value in the matching case expression.If no matches are found, the result of the `switch` is the value within the default case.

# Generating Field Values

In certain forms, you might want to first display a set of abstract derived fields to the user, then generate a different set of concrete resource account attribute values when the form is submitted. This is known as *form expansion*. An `<Expansion>` element is typically used in hidden fields that depend on editable fields in the form. One purpose of the `<Expansion>` element is to convert data that is familiar and readable to an end-user into data that is recognized by a resource. For example, a user can see a manager's full name in the form, but the system receives a unique ID that it recognizes as belonging to a manager.

The following example shows a field definition that uses conditional logic to convert the value derived for the `location` field in the previous example back into a three-letter abbreviation.

```
<Field name='accounts[Oracle].locCode'>
   <Expansion>
      <switch>
         <ref>location</ref>
         <case>
            <s>Austin</s>
            <s>AUS</s>
         </case>
         <case>
            <s>Houston</s>
            <s>HOU</s>
         </case>
         <case>
            <s>Dallas</s>
            <s>DAL</s>
         </case>
      </switch>
   </Expansion>
</Field>
```

The `<Expansion>` element is part of the Form XML language. The `<Expansion>` element can contain an expression. When this field is processed, the expression in the `<Expansion>` element is evaluated to determine the value of the field.

In this example, it performs the reverse of the mapping performed by the `location` field. This field is also hidden by the absence of an assigned `Display` class. This lack of `Display` class prevents the field from being displayed in the form, but the field is still considered to be an active part of the form and will generate values for resource attributes through its `<Expansion>` expression.

**Note** For all forms except the User view, `Expansion` rules are run whenever the page is recalculated or the form is saved. For the User view, an `<Expansion>` tag runs when the userform is first loaded as well.

# Workflow Transition Conditions

When defining a workflow process, you must specify the rules by which control passes from one workflow activity to another. A path between two activities is called a *transition*. A rule that governs the use of the transition is called a *transition condition*.

For example, consider the following activity definition:

```
<Activity name='Check Results'>

   <Transition to='Log Errors'>
      <gt> <ref>ERROR_COUNT</ref> <i>0</i> </gt>
   </Transition>

   <Transition to='end'/>

</Activity>
```

This activity defines two distinct transitions to separate activities: an activity named `Log Errors` and another named `end`. When workflow processes this activity, it will take the first transition for which the transition condition returns true.

In this example, the first transition has a condition that tests the value of the variable `ERROR_COUNT` to see if it is greater than zero. That transition is taken only if there is a positive error count. The second transition has no condition, and consequently will always be taken if the first transition condition is false.

## Workflow Actions

A workflow activity can perform one or more *actions*. One possible action is the evaluation of an XPRESS expression, as shown in the example below.

```
<Activity name='Increment Counter'>
   <Action>
      <expression>
         <set name='counter'>
            <add> <ref>counter</ref> <i>1</i> </add>
         </set>
      </expression>
   </Action>

   <Transition to='Next'/>

</Activity>
```

When a workflow action is implemented in XPRESS, an XPRESS expression is wrapped in an `expression` element that is then placed within an `Action` element. In this example, the expression references the current value of a variable named *counter*, adds one to that value, then assigns the incremented value to the variable with the same name.

## Workflow Actions Calling Java Methods

Complex workflow actions can be implemented in Java. Typical examples of complex workflow actions include storing data in a relational database or sending a message to a help desk system. These Java classes can be integrated with workflow using XPRESS.

```
<Activity name='Log Status'>

   <Action>
      <expression>
         <invoke name='logStatus'
            class='custom.OracleStatusLog'>
            <ref>accountId</ref>
            <ref>email</ref>
            <ref>status</ref>
         </invoke>
      </expression>
   </Action>
   <Transition to='Next'/>

</Activity>
```

In this example, the XPRESS `invoke` function is used to call a static method named `logStatus`, which is defined in the custom Java class `custom.OracleStatusLog`. Three arguments are passed to this method, the values of which are taken from workflow variables.

In these types of cases, the primary computation is performed in the Java class, while XPRESS acts to integrate the class into the workflow.

## Testing Expressions

Testing expressions involves two steps:

- Checking XML Syntax with the lh Command
- Tracing XPRESS Evaluation

## Checking Expression Syntax with lh Command

To check the XML syntax of expressions without actually evaluating their logic:

1. Confirm that you have `%WSHOME%\bin` in your `PATH` environment variable. (For information on changing environment variables to work with Identity Manager, see the section of *Identity Manager Installation* that describes using command-line tools.)

   If `%WSHOME%\bin` is not in your path, then you must change to `%WSHOME%\bin` before you can run the tools.

2. From the command line, enter `lh xmlparse <xpress_file>` where *`xpress_file`* represents the name of the file that contains the XML you want to test. This command parses the file for XML correctness and displays error messages in the console.

**Note** Consider putting `%WSHOME%\bin` in your `PATH` environment variable. This will permit you to use whichever directory you are currently in as your working directory. This will also allow you to run the Identity Manager `lh` command from any current working directory.

## Tracing XPRESS Evaluation

Once you have written and successfully stored an expression in the repository, you can turn on XPRESS tracing to determine if the expression is functioning correctly. XPRESS trace messages are sent to the standard output device. Since XPRESS is typically evaluated within the application server, the trace messages are sent to the console window or log file that was active when the application server was started.

There are two forms of XPRESS tracing:

- **Global trace**. When global trace is enabled, all XPRESS expressions are traced.
- **Block-level trace**. When block level tracing is used, only expressions within designated blocks are traced. Block tracing can be set only within a field element in a form or within an expression in a workflow.

Typically, block-level tracing is preferable since it reduces the amount of trace output, which is then easier to analyze.

### Enabling Tracing

To enable global trace, set a `Waveset.properties` file entry named `xpress.trace` to the value `true`. If you change the `Waveset.properties` file while the application server is running, you must either restart the application server, or go to the Debug Page and click **Reload Properties**.

To perform block-level trace, wrap the expressions you want to trace in a `<block>` expression and include the attribute `trace='true'` in the block start tag.

### Valid Examples

```
<block trace='true'>
   <invoke name='getTime' class='java.util.Date'/>
</block>

or

<Default>
   <block trace = 'true'>
      <ref>global.accountId</ref>
   </block>
</Default>
```

### Invalid Examples

Do not use the `<block>` element in the following ways.

```
<block trace='true'>
   <Field name ='field1'>
…
   </Field>
</block>

or

<Field name='Field2'>
   <block trace='true'>
      <Default>
         <ref>global.accounts</ref>
      </Default>
   </block>
</Field>
```

The trace messages include the names of the functions, the values of each argument, and the return values.

To turn tracing off for XPRESS, set the `xpress.trace` value to `false`, and reload the `Waveset.properties` file.

# Functions

Identity Manager ships with a library of XPRESS functions that can be used in expressions. These functions are classified into the following categories:

- Value constructors
- Arithmetic expressions
- Logical expressions
- String manipulation expressions
- List manipulation expressions
- Conditional, iteration, and block expressions
- Variable and function definition expressions
- Object access expressions
- Java and JavaScript access expressions
- Debugging and testing expressions

## Value Constructor Expressions

In XPRESS, literal values are written as text contained with an XML element. The element name is the name of a function, and the literal text is the argument to the function. The following functions are provided for constructing simple atomic data types.

### i Function

Constructs an integer value. The function takes one argument, which must be literal text. The text should contain only numeric digits and can be optionally preceded by a plus or minus.

#### Example 1

```
<i>0</i>
```

#### Example 2

```
<i>42</i>
```

#### Example 3

```
<i>-1234</i>
```

## list Function

Builds a value of type *list* by evaluating each of the argument expressions and concatenating the return values. The expression can take multiple arguments.

### Example

```
<list>
   <s>apples</s>
   <s>oranges</s>
   <s>wiper blades</s>
</list>
```

## null Function

Constructs a null value.

### Example 1

```
<null/>
```

### Example 2

```
<null></null>
```

## s Function

Constructs a string value. The function takes one argument, which must be literal text. (Length is constrained only by the amount of contiguous memory available in your Java environment.)

### Example

```
<s>Now is the time</s>
```

# Arithmetic Expressions

Use the following functions to perform arithmetic processing within expressions.

## add Function

Performs integer summation over the values of all arguments. Arguments that are not integers are coerced to integers.

### Example

The following expression results in an integer (42).

```
<add> <i>40</i> <i>1</i> <s>1</s> </add>
```

## div Function

Performs successive integer division over the values of all arguments. Arguments that are not integers are coerced to integers.

### Example

The following expression results in an integer (42).

```
<div> <i>84</i> <i>2</i> </div>
```

## mod Function

Performs successive integer modulo over the values of all arguments. Arguments are coerced to integers. Arguments of type `null` are ignored.

### Example

The following expression results in an integer (42).

```
<mod> <i>142</i> <i>100</i> </mod>
```

### mult Function

Performs successive integer multiplication over the values of all arguments. Arguments that are not integers are coerced to integers.

#### Example

The following expression results in an integer (42).

```
<mult> <i>7</i> <i>3</i> <i>2</i> </mult>
```

### sub Function

Performs successive integer subtraction over the values of all arguments. Arguments that are not integers are coerced to integers.

#### Example

The following expression results in an integer (42).

```
<sub> <i>50</i> <i>6</i> <i>2</i> </sub>
```

## Logical Expressions

Use the following functions to perform logical operations within expressions. Most logical functions return `1` and `0` to indicate true or false. The exceptions are `cmp`, `ncmp`, `and`, and `or`.

### and Function

Takes any number of arguments and returns zero if any argument values are logically false. If all arguments are logically true, the function returns the value of the last argument. Zero (`<i>0</i>` or `<s>0</s>`) and `<null>` are considered logically false.

#### Example 1

The following expression returns zero.

```
<and> <i>42</i> <s>cat</s> <i>null</i> </and>
```

#### Example 2

The following expression returns `cat`.

```
<and> <i>42</i> <s>cat</s> </and>
```

## cmp Function

Compares two string values. You can use this function to sort a list of strings

The function returns:

- *negative number* – value of the first argument is lexically less than the second.
- *positive number* – first argument is lexically greater than the second
- `0` (zero) – arguments are equal

Arguments are coerced to strings, if necessary.

### Example 1

The following expression returns `-1`.

```
<cmp>
   <i>20</i>
   <i>100</i>
</cmp>
```

### Example 2

The following expression returns -16. This expression returns a number that indicates the difference between the letters `r` and `b` when presented in alphabetical order. Since there are 16 letters between the letters `b` and `r`, when `bob` is compared to `ray`, the value is -16. Alternatively, if `r` were compared to `b`, the value returned would be 16.

```
<cmp>
   <s>bob</s>
   <s>ray</s>
</cmp>
```

### Example 3

The following expression returns `0` (zero).

```
<cmp>
   <s>daryl</s>
   <s>daryl</s>
</cmp>
```

## eq Function

Performs an equality test. The function can take multiple arguments, although typically it has only two. The data type of the first argument defines how the equality test is performed. If the first argument is of type:

- `string` – all subsequent arguments are coerced to strings, and string comparison is performed
- `integer` – all subsequent arguments are coerced to integers, and numeric comparison is performed
- `object` – all subsequent arguments must be of type object, and they must reference the same object

This function returns:

`0` – statement is logically false

`1` – statement is logically true

### Example

```
<eq> <ref>role</ref> <s>engineering</s> </eq>
```

## gt Function

Takes two arguments.

This function returns:

- `0` – the first argument is numerically less than or equal to the second
- `1` – the first argument is numerically greater than the second

### Example

```
<gt>
   <ref>age</ref>
   <i>42</i>
</gt>
```

## gte Function

Takes two arguments.

This function returns:

- `0` – first argument is less than the second
- `1` – the first argument is numerically greater than or equal to the second

### Example

The following expression returns 1.

```
<gte>
   <i>10</i>
   <i>5</i>
</gte>
```

## isfalse Function

Often used when referencing Boolean values that are represented with the strings `true` and `false` rather than the numbers 0 and 1. Takes two arguments.

This function returns:

- `0` – the argument is logically true (non-zero and non-null) and not the string `false`
- `1` – the argument is logically false or returns the string `false`.

### Example

The following expression returns 1.

```
<isFalse>
   <s>false</s>
</isFalse>
```

## isNull Function

Takes one argument.

This function returns:

- `0` – statement is non-null
- `1` – statement is null.

### Example 1

The following expression returns `1`.

```
<isnull> <null/> </isnull>
```

### Example 2

The following expression returns `0`.

```
<isnull> <i>0</i> </isnull>
```

## isTrue Function

Often used when referencing Boolean values that are represented with the strings `true` and `false` rather than the numbers 0 and 1. Takes two arguments.

This function returns:

- `0` – the argument is logically false (non-zero and non-null) or if the argument is the string `false`
- `1` – the argument is logically true and not the string `false`.

### Example

The following expression returns 0.

```
<isTrue>
   <s>false</s>
</isTrue>
```

## lt Function

Takes two arguments.

This function returns:

- `0` – first argument is numerically greater than or equal to the second
- `1` – first argument is numerically less than the second

### Example 1

The following expression returns `0` (zero).

```
<lt>
   <i>10</i>
   <i>5</i>
</lt>
```

### Example 2

The following expression returns `1`.

```
<lt>
   <i>5</i>
   <i>10</i>
</lt>
```

## lte Function

Takes two arguments.

This function returns:

- `0` – first argument is numerically greater than the second
- `1` – first argument is numerically less than or equal to the second

### Example

```
<lte>
   <ref>age</ref>
   <i>42</i>
</lte>
```

## ncmp Function

Performs case-insensitive comparison of two string values.

This function returns:

- *negative number* – indicates that the value of the first argument is lexically less than the second
- *positive number* – indicates that the first argument is lexically greater than the second
- `0` (zero) – indicates that the arguments are equal

Arguments are coerced to strings, if necessary.

### Example

The following expression returns 0.

```
<ncmp>
   <s>Daryl</s>
   <s>daryl</s>
</ncmp>
```

## neq Function

Performs an inequality test. Its behavior is simply the negation of the equality test performed by the `eq` function.

This function returns:

- `0` – the two arguments are equal
- `1` – the two arguments are not equal

### Example

```
<neq>
   <ref>role</ref>
   <s>management</s>
</neq>
```

## not Function

Reverses the logic of the nested expression.

This function returns:

- `1` – the value of the argument is logically false
- `0` – argument is logically true

### Example

The following example returns `0`.

```
<not> <eq> <i>42</i> <i>24</i> </eq> </not>
```

## or Function

Takes multiple arguments.

This function returns:

`null` – all arguments are logically false

Value of the first argument expression that results in a logically true value

### Example 1

The following expression returns null, which is logically false.

```
<or> <i>0</i> <i>0</i> </or>
```

### Example 2

The following expression returns the string `cat`, which is also logically true.

```
<or> <i>0</i> <s>cat</s> </or>
```

### notNull Function

Takes one argument

This function returns:

`0` – null argument

`1` – non-null argument

#### Example 1

The value of the following expression is 1 if the `firstname` has been set or 0 (zero) if `firstname` is null.

```
<notnull>
   <ref>firstname</ref>
</notnull>
```

#### Example 2

The value of the following expression is 0 since the value is null.

```
<notnull><null/></notnull>
```

## String Manipulation Expressions

Use the following functions to perform string manipulation within expressions.

### indexOf Function

Returns the position of a string within another string.

#### Example

The following function returns 3.

```
<indexOf>
   <s>Austin City Limits</s>
   <s>tin</s>
</indexOf>
```

## concat Function

Concatenates two or more string values.

### Example

The following expression returns `<s>Now is the time</s>`.

```
<concat>
   <s>Now </s><s>is </s><s>the </s><s>time</s>
</concat>
```

## downcase Function

Takes a single argument, which is coerced to a string. It returns a copy of the argument with all upper case letters converted to lower case.

### Example

The following expression returns `<s>abc</s>`.

```
<downcase><s>ABC</s></downcase>
```

## ltrim Function

Takes a single argument, which is coerced to a string.

It returns a copy of the argument with the leading white space removed.

### Example

The following expression returns `<s>hello</s>`.

```
<ltrim><s>   hello</s></ltrim>
```

## match Function

Tests for a substring within a string. Similar to the `indexOf` function, this function is more convenient to use in conditional expressions. If the substring starts at the first position in the string, `indexOf` would return 0, which is logically false.

This function returns:

0 – substring is not found

1 – substring is found

### Example

The following expression returns 1.

```
<match>
   <s>Austin City Limits</s>
   <s>tin</s>
</match>
```

## pad Function

Pads a string with spaces so that it reaches a desired length.

*`first argument`* – the string to `pad`

*second argument* – desired length

*third argument* – (optional) specifies the `pad` character, which by default is a space

### Example

The following expression results in `<s> email  </s>`

```
<pad>
   <s> email</s>
   <i>10</i>
</pad>
```

## rtrim Function

Takes a single argument, which is coerced to a string. It returns a copy of the argument with the trailing white space removed.

### Example

This example returns 0 (zero).

```
<cmp>
   <s>hello</s><rtrim><s>hello   </s></rtrim>
</cmp>
```

## split Function

Splits a string into a list of strings.

*first argument* – string to be split

*second argument* – a set of one or more string delimiters. Each character in this string will cause a break.

A list is created that contains each substring between delimiters.

### Example 1

```
<split>
   <s>Austin City Limits</s>
   <s> </s>
</split>
```

This expression returns the following list.

```
<list>
   <s>Austin</s>
   <s>City</s>
   <s>Limits</s>
</list>
```

### Example 2

The following expression uses multiple delimiters.

```
<split>
   <s>(512)338-1818</s>
   <s>()-</s>
</split>
```

This expression returns the following list.

```
<list>
   <s>512</s>
   <s>338</s>
   <s>1818</s>
</list>
```

## substr Function

Extracts ranges of characters from a string.

This function takes two forms:

- *start* and *length* are specified as arguments (child nodes of the `substr` element).
- *start* and *length* are specified as attributes of the substr node *s* for *start* and *l* for *length*.

For example, these two invocations are equivalent:

```
<substr>
   <s>Hello World</s>
     <i>3</i>
     <i>4</i>
<substr>
```

and

```
<substr s='3' l='4'>
   <s>Hello World</s>
<substr>
```

Both functions return the string `lo W`.

```
<block>
   <substr s='3' l='4'>
      <s>Hello World</s> --> Hello World
   </substr> --> lo W
</block> --> lo W
```

The *start* and *length* parameters are optional. If the *start* argument is missing, either because only the string is specified as a child of the substr node as in

```
<substr>
   <s>Hello World</s>
<substr>
```

and the attribute *s* is also missing from the substr node, the start is assumed to be the beginning of the string. In other words, its value is zero if not specified explicitly.

*first argument* – string

*second argument* – starting position

*third argument* – number of characters to retrieve

## Examples

The following expression returns <s>Now</s>.

```
<substr>
   <s>Now is the time</s>
   <i>0</i>
   <i>3</i>
</substr>
```

In the following example, the *start* attribute is missing, but is assumed to be 0:

```
<block>
   <substr l='4'>
       <s>Hello World</s> --> Hello World
   </substr> --> Hell
</block> --> Hell
```

The *length* argument is also optional. A missing *length* argument causes the function to extract the rest of the string. *length* can be unspecified when only the *string* and *start* arguments are specified a child nodes of substr such as:

```
<substr>
   <s>Hello World</s>
       <i>3</i>
<substr>
```

or when the *l* attribute is missing from the substr node like. Note that the *length* argument is unspecified below, but the rest of the string starting from this start is returned:

```
<block>
   <substr s='3'>
      <s>Hello World</s> --> Hello World
   </substr> --> lo World
```

```
</block> --> lo World
```

### trim Function

Takes a single argument, which is coerced to a string.

It returns a copy of the argument with the leading and trailing white space removed.

#### Example

The following expression returns `<s>hello</s>`.

```
<trim><s> hello </s></trim>
```

### upcase Function

Takes a single argument, which is coerced to a string.

It returns a copy of the argument with all lower case letters converted to upper case.

#### Example

The following expression returns `<s>ABC</s>`.

```
<upcase><s>abc</s></upcase>
```

## List Manipulation Expressions

Most list manipulation functions have two forms depending upon whether the `name` attribute is included in the function element:

- If included in the function element, the name is expected to resolve to a variable containing a list. In this case, the referenced variable is destructively modified. The following example modifies the list stored in the `someList` variable and adds two elements:

```
<append name='someList'>
   <s>Hello</s>
   <s>World</s>
</append>
```

If the name is not included in the function element, a new list is constructed. In the following example, a new list is created by combining the elements of the list stored in the `someList` variable with two additional elements. The value of the `someList` variable is not modified.

```
<append>
   <ref>someList</ref>
   <s>Hello</s>
   <s>World</s>
</append>
```

Use the following functions to manipulate list elements.

## append Function

Appends a value to a list. The argument list takes one of two forms depending on the presence of the `name` attribute.If `name` is not specified, then the first argument must be a list and the remaining arguments are elements to append to that list. A copy of the list is returned, the original list is not modified. If the *name* argument used, then all arguments are considered objects to be appended to the list contained in the variable with that name. The list is modified without being copied.

### Example 1

The following expression makes a copy of the list contained in the variable *`srclist`* then appends one element.

```
<append>
   <ref>srclist</ref>
   <s>oranges</s>
</append>
```

### Example 2

The following expression modifies an existing list by appending a value.

```
<set name= 'somelist'>
  <List>
    <s>We</s>
    <s>say</s>
  </List>
</set>
<append name= 'somelist'>
  <s>Hello</s>
  <s>World</s>
</append>
<ref>someList</ref>
```

## appendAll Function

Merges the elements in multiple lists. If the `name` attribute is specified, an existing list is modified. Otherwise, a new list is created.

### Example 1

The following expression creates a new list by combining the elements in `srclist` with three additional elements.

```
<appendAll>
   <ref>srclist</ref>
   <list>
      <s>apples</s>
      <s>oranges</s>
      <s>peaches</s>
   </list>
</appendAll>
```

### Example 2

The following expression adds three elements to the list stored in the variable `srclist`.

```
<appendAll name='srclist'>
   <list>
      <s>apples</s>
      <s>oranges</s>
      <s>peaches</s>
   </list>
</appendAll>
```

## contains Function

*first argument* – list

*second argument* – any object to search for in the list

This function returns:

1 -- list contains a given value

### Example

The following expression returns 1.

```
<contains>
   <list>
      <s>apples</s>
      <s>oranges</s>
   </list>
   <s>apples</s>
</contains>
```

## containsAll Function

Takes two list arguments.

This function returns:

1 -- the list contains all elements contained in another list

0 (zero) -- the list does not contain all elements contained in the second list

### Example

The following expression returns 0.

```
<containsAll>
   <ref>fruitlist</ref>
   <list>
      <s>oranges</s>
      <s>wiper blades</s>
   </list>
</containsAll>
```

## containsAny Function

*first argument* – list to be searched

*second argument* – an element or a list of elements to search for in the first list

This function returns:

1 -- first list contains any elements that are contained in a second list.

0 (zero) -- first list does not contain any elements that are contained in a second list.

### Example

The following expression returns 1.

```
<containsAny>
   <ref>fruitlist</ref>
   <list>
      <s>oranges</s>
      <s>wiper blades</s>
   </list>
</containsAny>
```

## filterdup Function

Filters duplicate elements from a list. Given a list, it returns a new list in which duplicate entries have been removed.

### Example

```
<filterdup>
   <list>
      <s>apples</s>
      <s>oranges</s>
      <s>apples</s>
   </list>
</filterdup>
```

This expression returns the following list.

```
<list>
   <s>apples</s>
   <s>oranges</s>
</list>
```

## filternull Function

Filters null elements from a list.

This function returns a single list removing all null elements (when given one list).

### Example

```
<filternull>
   <list>
      <s>apples</s>
      <null>
         <s>oranges</s>
      <null/>
   </list>
</filternull>
```

This expression returns the following list.

```
<list>
   <s>apples</s>
   <s>oranges</s>
</list>
```

## get Function

Retrieves the value of the nth element in the list. The list indexes starts count from zero (0). Arguments are a list and an integer.

### Example

```
<get>
   <list>
      <s>apples</s>
      <s>oranges</s>
   </list>
   <i>1</i>
</get>
```

This expression returns `<s>oranges</s>`

## indexOf Function

*first argument* – a list value to search

*second argument* – value for which to search

This function returns either the ordinal position of a list element that matches a given value or `-1` (the given value is not in the list).

### Example

The following expression returns `1`.

```
<indexOf>
   <list>
      <s>apples</s>
      <s>oranges</s>
   </list>
   <s>oranges</s>
</indexOf>
```

## insert Function

Inserts a value into the list. Elements following the insertion index down are shifted to make room for the new element.

*first argument* – a list to which an element is inserted

*second argument* – integer specifying position in the list at which to insert the new element

*third argument* – value to insert into the list

### Example

```
<insert>
   <list>
      <s>apples</s>
      <s>oranges</s>
   </list>
   <i>1</i>
   <s>wiper blades</s>
</insert>
```

This expression returns the following list.

```
<list>
   <s>apples</s>
   <s>wiper blades</s>
   <s>oranges</s>
</list>
```

## length Function

Returns the number of elements in the list. You can also use this function to return the length of a string.

*first argument* – list or string

### Example 1

The following expression returns `2`.

```
<length>
   <list>
      <s>apples</s>
      <s>oranges</s>
   </list>
</length>
```

### Example 2

```
<length>
   <s>Hello world!</s>
</length>
```

This expression returns a value of `11`.

## remove Function

Removes one or more elements from a list. The argument list takes one of two forms depending on the presence of the `name` attribute. If `name` is not specified, then the first argument must be a list and the remaining arguments are elements that are removed from that list. A copy of the list is returned. (The original list is not modified.) If the *name* argument used, then all arguments are considered objects to be removed from the list contained in the variable with that name. The list is modified without being copied.

### Example 1

The following expression makes a copy of the list contained in the variable *srclist*, then removes one element and returns the copy of the list.

```
<remove>
   <ref>srclist</ref>
   <s>oranges</s>
</remove>
```

### Example 2

The following expression modifies an existing list by removing a value.

```
<set name= 'somelist'>
  <List>
    <s>We</s>
    <s>say</s>
  </List>
</set>
<remove name= 'somelist'>
  <s>say</s>
  <s>say</s>
</remove>
<ref>someList</ref>
```

## removeAll Function

Removes all elements contained in one list from another list. If the `name` attribute is specified, an existing list is modified. Otherwise, a new list is created.

### Example 1

The following expression creates a new list by removing the elements in `srclist` along with three additional elements.

```
<removeAll>
   <ref>srclist</ref>
   <list>
      <s>apples</s>
      <s>oranges</s>
      <s>peaches</s>
   </list>
</removeAll>
```

### Example 2

The following expression removes three elements in the list stored in the variable `srclist`.

```
<removeAll name='srclist'>
   <list>
      <s>apples</s>
      <s>oranges</s>
      <s>peaches</s>
   </list>
</removeAll>
```

This expression results in the following list.

```
<list>
   <s>wiper blades</s>
</list>
```

## set Function

Assigns a value into a specified position in a list, overwriting its current value. If necessary, the list is extended to contain the indexed element. New elements created during list extension will be null.

*first argument* – list

*second argument* – integer specifying position in the list at which to insert the new element, starting with zero.

*third argument* – element

### Example 1

```
<set>
   <list>
      <s>apples</s>
      <s>oranges</s>
      <s>wiper blades</s>
   </list>
   <i>2</i>
   <s>bassoons</s>
</set>
```

This expression results in the following list and returns null.

```
<list>
   <s>apples</s>
   <s>oranges</s>
   <s>bassoons</s>
</list>
```

## Example 2

```
<set>
   <list>
      <s>apples</s>
      <s>oranges</s>
      <s>wiper blades</s>
   </list>
   <i>5</i>
   <s>bassoons</s>
</set>
```

This expression results in the following list and returns null.

```
<list>
   <s>apples</s>
   <s>oranges</s>
   <s>wiper</>
   </null>
   </null>
   <s>bassoons</s>
</list>
```

# Conditional, Iteration, and Block Expressions

Use these functions to perform conditional and block processing within expressions.

## block Function

Groups more than one expression into a single expression. The value of the `block` function is the value of its last argument.

**Note** The `<set>` function does not return a value. If the last line in a `block` statement involves a `set` operation, the block statement will not return a value. If you want the block statement to return the value of a variable, use `<ref>`***variable_name***`</ref>` on the last line of the block statement.

### Example

```
<block>
   <s>Hello there!</s>
   <add> <i>100</i> <i>2</i> </add>
   <i>42</i>
</block>
```

The block returns a value of `42`, the value of its last argument.

For an example of using block with a trace statement, see *Debugging and Testing Functions*.

## break Function

Forces early termination of an expression. A break can be used within the following expressions: `block`, `dolist`, `while`, and, `or`. The value of the `break` expression becomes the value of the containing expression. The break can cause the termination of several levels of expression when the optional block name is used.

### Example 1

The following expression contains a simple break terminating a loop.

```
<dolist name='el'>
<ref>list</ref>
   <cond><eq><ref>el</ref><s>000</s></eq>
      <break>
         <ref>el</ref>
      </break>
   </cond>
<null/>
</dolist>
```

In this example, the `dolist` function iterates over the elements of a list looking for value 000. The value of the `dolist` function is a list formed by concatenating the values that are returned by the last subexpression in each iteration.

### Example 2

The following expression demonstrates the use of a block name to break through more than one level.

```
<block name='outer block'>
  <dolist name='el'>
    <ref>listOfLists</ref>
    <dolist name='el2'>
      <ref>el</ref>
      <cond><eq><ref>el</ref><s>000</s></eq>
        <break name='outer block'>
          <ref>el</ref>
        </break>
      </cond>
    </dolist>
    <null/>
  </dolist>
</block>
```

This is similar to the previous example except that there are two loops. The outer loop iterates over a list whose elements are themselves lists. The inner loop iterates over the element lists. When the value 000 is found, both loops are terminated by referencing the block name `outer block` in the `break` expression.

## cond Function

Provides a way to conditionally select the value of one of two expressions. It is similar to the ternary conditional operator `(a?b:c)` in C and Java.

### Example

The `cond` function allows three arguments. The first argument is called the `condition`. If the value of the condition is logically true, the value of the `cond` will be the value of the second argument. If the value of the condition is false, the value of the `cond` will be the value of the third argument. If the value of the condition is false, and the third argument not present, the value of the `cond` is null.

```
<cond>
   <gt>
      <ref>age</ref>
      <i>40</i>
   </gt>
   <s>old</s>
   <s>young</s>
</cond>
```

## dolist Function

Iterates over the elements of a list. The value of the `name` attribute will become the name of variable that can be referenced within the loop.

The value of this variable will be the value of successive list elements.

The first subexpression returns the list over which to loop. The remaining sub expressions are repeated once for each element in the list.

The value of the `dolist` function is a list formed by concatenating the values returned by the last subexpression in each iteration.

### Example 1

The following expression creates a list called `subset`, which contains the subset of elements in `srclist` that exceed 10.

```
<set name='subset'>
   <dolist name='el'>
     <cond>
       <gt>
         <ref>el</ref>
         <i>10</i>
       </gt>
       <ref>el</ref>
     </cond>
   </dolist>
</set>
```

### Example 2

The following expression returns `apples`.

```
<switch>
   <s>A</s>
   <case default='true'>
      <s>unknown</s>
   </case>
   <case>
```

```
      <s>A</s>
      <s>apples</s>
   </case>
   <case>
      <s>B</s>
      <s>oranges</s>
   </case>
</switch>
```

This expression returns the value of counter, which is 0.

## select Function

Returns the first non-null value in a list. Use this function when you need to obtain the correct context from, for example, a workflow.

### Example

```
<select>
   <ref>:display.session</ref>
   <ref>context</ref>
</select>
```

## while Function

Repeats a set of expressions until a condition is met. The first subexpression is called the `conditional` and will be evaluated each time through the loop. The loop terminates when the conditional is logically false. The value of the `while` expression is the value of the last expression in the loop during the last iteration.

### Example

The following expression returns null.

```
<while>
   <gt>
      <ref>counter</ref>
      <i>0</i>
   </gt>

   <set name='counter'>
      <sub> <ref>counter</ref>
         <i>1</i>
      </sub>
   </set>
</while>
```

# Variables and Function Definition Expressions

Use the following functions to reference and define variables and functions within expressions.

## ref Function

Reference the value of a variable. The variable can either be an external variable supported by the host application or an internal variable defined with `<defvar>`.

### Example

```
<ref>waveset.role</ref>
```

## defvar Function

Defines a new variable. The variable can then be referenced by any expression within and below the containing expression in which the variable was defined. The variable must be given a name using the XML attribute `name`.

A `defvar` statement should not reference itself. If it does, it will cause a loop.

To change the value of an existing variable, see the description of the `set` function in this section.

**Note** Avoid the following constructions:

```
<defvar name='fullname'>
   <ref>fullname</ref>
</defvar>
```

or

```
<defvar name='counter'/>
   <add><ref>counter</ref>
      <i>0</i>
   </add>
</defvar>
```

### Example 1

The following expression defines a variable and initializes its value to a list of two elements.

```
<defvar name='theList'>
  <list>
     <s>apples</s>
     <s>oranges</s>
```

```
  </list>
</defvar>
```

### Example 2

The following expression defines a variable and initializes its value to the integer zero.

```
<defvar name='counter'>
   <i>0</i>
</defvar>
```

## defarg Function

Used to define an argument within a function defined with `<defun>`. Arguments are similar to variables, but they must be defined in the order in which arguments are passed to the function.

### Example

```
<defarg name='arg1'/>
<defarg name='arg2'/>
```

## defun Function

Defines a new function. The `<defarg>` function must be used to declare the arguments to a function. Use the `<call>` function to execute the function. Functions are typically defined within forms.

### Example

```
<defun name='add100'>
   <defarg name='input'/>
   <add>
      <ref>input</ref>
      <i>100</i>
   </add>
</defun>
```

## call Function

Calls a user-defined function. The arguments to call are assigned to arguments with `<defarg>` in the so-called function. The order of the `call` arguments must match the order of the `<defarg>`s. In previous releases, the call function could be used to call rules. Now, use the rule function for that purpose.

### Example

The following expression returns `142`.

```
<call name='add100'>
   <i>42</i>
</call>
```

## rule Function

Calls a rule. The arguments to rule are passed by name using the argument element. The value of an argument can be specified with the value attribute if it is a simple string. The argument value can also be calculated with an expression by omitting the value attribute and instead writing an expression within the body of the argument element.

A `<rule>` element can also call another rule that dynamically calculate the name of another rule to call.

For more information on creating or calling rules in forms and workflows, see the chapter titled *Rules*.

### Examples

The following expression returns the employee ID of the designated user.

```
<rule name='getEmployeeId'>
   <argument name='accountId' value='maurelius'/>
</rule>

<rule name='getEmployeeId'>
   <argument name='accountId'>
      <ref>username</ref>
   </argument>
</rule>
```

The following expression calls another rule that calculates the returned value.

```
<rule>
   <cond>
     <eq><ref>var2</ref><s>specialCase</s></eq>
     <s>Rule2</s>
     <s>Rule1</s>
   </cond>
   <argument name='arg1'>
     <ref>variable</ref>
   </argument>
</rule>
```

## set Function

Can be used to change the value of an existing variable. The behavior of this function depends upon the type of the first argument.

The `set` function takes one of four forms:

- If the `name` attribute is specified, the function modifies the value of the variable with that name. The argument to the `set` function will become the new value for the variable.
- If the `name` attribute is not specified, the behavior is defined by the type of the first argument. If the type of the first argument is a string, the string is assumed to be the name of a variable, and the variable value is changed to the value of the second `set` argument. This can be used to compute the name of the variable that is to be modified.
- If the first argument is a `list`, the second argument is expected to be an integer list index, and the third argument is a value to be placed in the list. If the list size is greater than the index, the current list element at that index is replaced. If the index is greater than or equal to the list size, the list is extended and the new value is added.
- If the first argument is a Map, the second argument is expected to be a map key and the third argument is a map value. An entry is placed in the map with that key and value.

### Example

In the following example, the variable ***var_name*** is set to the first argument.

```
<set name='var_name'>
  <s>new value</s>
</set>
```

# Object Manipulation Expressions

Use the following functions to manipulate arbitrary object values within expressions.

## get Function

The function can be used to access attributes of an object or to extract an element from a list.

*first argument* – an object.

*second argument* – name of accessor function to call on object; for example, if the method `getName` exists, this parameter would be `<s>name</s>`.

The function behaves differently if the first argument is a list. If the first argument is a list, then the second argument is an integer list index. The element at that index is returned.

### Example

This expression returns a string that is the name of the currently assigned role for the user.

```
<get>
   <ref>userView</ref>
   <s>role</s>
</get>
```

This expression is equivalent to call `userView.getRole()` in Java code.

## getObj Function

This function is used to obtain an object from the Identity Manager repository. It is most commonly used to access user objects, whose type name would be `User`.

The arguments are concatenated together and expected to produce a string that follows the convention:

```
<typename>:<objectname>
```

where <*typename*> is the name of a repository type, and <*objectname*> is the name of an object of that type.

### Example

This expression retrieves the email address of a Identity Manager user in order to pass it to the notify workflow application.

```
<Action application='notify'>
   <Argument name='template' value='Provisioning Approval'/>
   <Argument name='toAddress'>
      <get>
         <getobj>
            <s>User:</s>
            <ref>APPROVER</ref>
         </getobj>
         <s>Email</s>
      </get>
   </Argument>
</Action>
```

## set Function

The behavior of this function depends upon the type of the first argument.

`list` – you can use the function to assign list elements.

`string` – the string is assumed to be the name of a variable, and the value of the variable is assigned.

*any other object* – the second argument must be the name of an attribute on that object, and the third argument must be the value to assign. Example

The following expression sets the value of the role attribute to engineering.

```
<set>
   <ref>userView</ref>
   <s>waveset.role</s>
   <s>engineering</s>
</set>
```

# Java and JavaScript Expressions

Use the following functions to call and manipulate Java classes or JavaScript functions from within expressions.

## invoke Function

Invokes a method on a Java object or class. To call a method on a Java object, the first argument specifies the Java object and the remaining arguments are passed as arguments to the method. The method name is specified with the `name` XML attribute. To call a static class method, the Java class name is specified with the `class` XML attribute. All arguments are passed as arguments to the class method.

To use this function, you must be familiar with the class and method names you want to call, the arguments they take, and the method's actions. This function is frequently used to call the following Identity Manager classes:

- FormUtil
- LighthouseContext
- WorkflowContext
- WavesetResult

For more information, see the available documentation for these classes.

## new Function

Creates an instance of a Java class. The class name is provided in the XML `class` attribute and must be fully qualified with the class package name.

You can also use this function to create a new object and return it as the value of an expression or rule without necessarily invoking methods on it.

### Example

```
<new class='java.util.Date'/>
```

## script Function

Encapsulates a fragment of JavaScript. When this expression is evaluated, the JavaScript interpreter is launched to process the script. The value of the expression is the value of the last JavaScript statement. Within the script, the object `env` can be used to access variables in the host application.

It is generally recommended that you avoid JavaScript in performance-critical expressions such as `<Disable>` expressions in forms. Short XPRESS expressions are easier to debug using the built-in tracing facilities. JavaScript is recommended for complex logic in workflow actions.

### Example

```
<script>
   var arg1 = env.get('arg1');
   arg1 + 100;
</script>


<script>
   importPackage(Packages.java.util);
   var cal Now = Calendar.getInstance();
   cal Now.getTime()
</script>
```

# Debugging and Testing Expressions

Enabling tracing can result in a large amount of trace data.

Use the following functions to enable expression trace or print text to help diagnose problems in an expression.

**Note** Globally enabling trace may result in a large amount of trace data being printed. If it is usually better to enable trace at the block level by setting the trace attribute of the block element to true.

## trace Function

Enables or disables expression tracing. If the argument evaluates to `true`, tracing is enabled.

If tracing is enabled, it will go to standard output.

### Example 1

```
<trace><i>1</i></trace>
```

### Example 2

```
<trace><i>0</i></trace>
```

## print Function

Prints the value of each subexpression to standard output.

### Example

```
<print>
   <s>Ashley World!</s>
</print>
```

# Data Types

All functions return a value that has one of the data types listed in the following table.

| Data Type | Definition |
|---|---|
| `integer` | Represents a signed integral value. The precision of the value is at least 32 bits. |
| `list` | Represents ordered lists of other values. The values in a list are called *elements*. List elements can be null. A list lacking elements is not considered to have a null value. |
| `null` | Represents the absence of a value. A function might return null if it is called only for side effect, or if it cannot compute a meaningful value from the given arguments. The way a `null` value is handled depends on the function being passed a `null` argument. In general, a `null` value is considered to be logically false and is ignored in arithmetic expressions. |
| `object` | Represents references to arbitrary objects that are defined outside the XPRESS language. |
| `string` | Represents a string of characters. Since XML syntax is used, strings always use the Unicode character set. A string value can contain no characters. Such a string is considered *empty*, but it is not null. |

Some functions treat the values of their arguments as being logically true or false. XPRESS does not use a Boolean data type. Instead, a value of null or an integer value of zero is considered false. Any other value is considered true.

Logical functions such as `eq` that return a Boolean value will return the integer zero to represent false and the integer 1 to represent true.

# 7 XML Object Language

The *XML Object Language* is a collection of XML elements that you can use to represent common Java objects such as strings, lists, and maps.

This chapter:

- Introduces XML Object language
- Compares XML Object language with XPRESS language
- Discusses using XML object language to create lists

## Introduction

XML Objects are often used in forms, but can also be used in workflows and rules. One common use is to create a list of allowed values for a `Select` or `MultiSelect` field in a form, as exemplified below.

**Example**

```
<Field name='global.state'>
    <Display class='Select'>
       <Property name='title' value='State'/>
       <Property name='allowedValues'>
         <List>
            <String>Alabama</String>
            <String>Alaska</String>
            <String>Arizona</String>
            <String>Arkansas</String>
            <String>California</String>
            <String>Washington</String>
            <String>Washington D.C.</String>
            <String>West Virginia</String>
            <String>Wisconsin</String>
            <String>Wyoming</String>
         </List>
       </Property>
    </Display>
</Field>
```

Elements in the XML Object language are similar to elements in the XPRESS language, but it is more efficient to use the XML Object language if the values are static.

These two languages differ primarily in that XML Object language does not allow the contents of an object to be computed with an expression. This restriction allows the system to construct the object more efficiently, which will result in faster processing if the object is large.

When defining lists with XML Object language, the list is created once when the form is read from the repository and reused thereafter. When defining lists with XPRESS, a new list is created every time the form is displayed.

# XML Object Language and Corresponding XPRESS

The following table lists several basic XML objects and the equivalent XPRESS expressions.

| XML Object Language | XPRESS Language |
|---|---|
| `<String>cat</String>` | `<s>cat</s>` |
| `<Integer>10</Integer>` | `<i>10</i>` |
| `<Boolean>true</Boolean>`<br>`<Boolean>false</Boolean` | `<i>1</i>`<br>`<i>0</i>` |
| `<null/>` | `<null/>` |
| `<Map>`<br>`  <MapEntry key='name' value='neko'/>`<br>`  <MapEntry key='ID' value='123'/>`<br>`</Map>` | `<map>`<br>`  <s>name</s>`<br>`  <s>neko</s>`<br>`  <s>ID</s>`<br>`  <i>123</i>`<br>`</map>` |
| `<List>`<br>`  <String>cat</String>`<br>`  <String>dog</String>`<br>`  <integer>673</Integer>`<br>`</List>` | `<list>`<br>`  <s>cat</s>`<br>`  <s>dog</s>`<br>`  <i>673</i>`<br>`</list>` |
| `<Long>123456789</Long>` | N/A |
| `<Date>20020911 09:15:00</Date>` | N/A |

You cannot use XPRESS statements within an XML object.

## Using XML Objects in XPRESS

XML objects can be used within XPRESS anywhere an expression is allowed. In the example below, a map is passed as an argument to an invoked method.

```
<invoke name='printTheMap'>
    <ref>mapPrinter</ref>
       <Map>
       </Map>
</invoke>
```

In releases prior to 2.0, XPRESS required that all XML Objects be wrapped in an `<o>` element. While this is no longer required, you may still encounter its use in older files containing XPRESS.

## When to Use XML Object Language Instead of XPRESS

Although both XML Object Language and XPRESS provide ways of representing lists in forms, XML Object syntax is more efficient than XPRESS if the list is long and contains static data. The list is built in memory once and it is reused every time it is referenced. In contrast, XPRESS list syntax is re-evaluated on every reference and a new list is created each time.

The XML object language is most typically used when creating lists of the information described in the following table.

| Type of Information Lists | Where Used |
|---|---|
| Machine names | forms |
| Business sites | forms |
| Approver names | workflow |

# Representing Lists in XML Object and XPRESS

Both XML Object Language and XPRESS provide ways of representing lists in forms.

## Using XPRESS to Represent a List

You use the `<list>` element when representing lists in XPRESS. The contents of the `<list>` element can be any XPRESS expression.

**Note** Use only the `<list>` XPRESS element in forms if the list must contain calculated elements. Using the `<list>` element can slow the execution of the form in which it is included. This degradation in performance is typically not noticeable unless the list contains many elements. It is permissible and common for forms to use <list>.

The following example uses the `<s>` string constants in the XPRESS list, but you can also use the `<invoke>` or `<concat>` elements to dynamically build the list elements.

**Example**

```
<list>
   <s>cat</s>
   <s>dog</s>
</list>
```

## Using XML Object Language to Represent a List

The XML Object language uses the `<List>` element to represent lists. The contents of the `<List>` element can be only other XML Objects. In the following example, the content of the `<List>` element are `<String>` elements.

**Example**

```
<List>
   <String>cat</String>
   <String>dog</String>
</List>
```

## Example Form Using Both Types of Syntax

The following form incorporates fields containing lists defined by both XML Object syntax and XPRESS.

### Example

```
<Form>
    <Field name='department'>
      <Display class='Select'>
        <Property name='allowedValues'>
          <List>
           <String>Engineering</String>
           <String>Marketing</String>
           <String>Sales</String>
          </List>
        </Property>
      </Display>
    </Field>

    <Field name='department2'>
      <Display class='Select'>
        <Property name='allowedValues'>
          <expression>
            <list>
              <s>Engineering</s>
              <s>Marketing</s>
              <s>Sales</s>
            </list>
          </expression>
        </Property>
      </Display>
    </Field>
</Form>
```

The `allowedValues` list in the department field is defined as a static list built with `<List>`. No matter how many times this form is used, only one list is created. In contrast, the `allowedValues` list in the `department2` field is defined with a `<list>` expression. A new list is created every time this form is used.

## Defining Map Objects with XML Object Syntax and XPRESS

You can use either the XML Object syntax or XPRESS to dynamically construct Map objects. Using the XPRESS `<map>` element is similar to using the XML Object language `<Map>` and `<MapEntry>` elements. These elements differ in that the contents of `<map>` can be calculated using expressions. In contrast, the `<Map>` element can only define static maps.

**Note** Maps are sometimes used as arguments to methods that are called with an <invoke> expression. For example, certain methods in the `FormUtil` class require maps as arguments. For more information on using map arguments with `FormUtil` methods, see the section titled *FormUtil Methods*.

## Using XPRESS to Represent a Map

The contents of the XPRESS `<map>` element are pairs of name/value expressions. The even-numbered expressions define map keys, and odd-numbered expressions define map values. If any key expression evaluates to null, the entry is ignored.

You can use the XPRESS `<map>` element to dynamically construct `java.util.HashMap` objects:

```
 <map>
   <name</s>
   <s>Jeff</s>
   <s>phone</s>
   <s>338-1818</s>
 </map>
```

## Using XML Object Syntax to Map Objects

You can use XML Object syntax to define map objects as follows:

```
<Map>
   <MapEntry key='name' value='Jeff'/>
   <MapEntry key='phone' value='338-1818'/>
</Map>
```

# 8 HTML Display Components

This chapter describes the Identity Manager HTML display component library. HTML display components are used when customizing forms. See *Forms* for a discussion of the larger topic of customizing forms.

## HTML Component

If you are designing forms, you will use the HTML components described in this section. To create a form, you can use the Identity Manager Form XML language (also called *forms*), to describe HTML display components. This language is then interpreted at runtime to build the necessary components. It allows new pages to be dynamically generated with little or no additional Java development, which greatly simplifies customization.

This section covers the following topics:

- What are HTML Components?
- Component classes
- Container classes
- Page processor requirements for HTML components

### What Are HTML Components?

HTML display components are instances of Java classes that generate a string of HTML text. Each display component has:

- A *class name* (defined in the field by the class attribute of the `Display` element). This name identifies the component class, which determines the component's fundamental behavior and defines the set of properties recognized by the component.
- One or more *properties* (defined in the field with `Property` elements). Properties further define field behavior and appearance.

#### Specifying Display Components

You can specify display components as follows:

```
<Field name='Name'>
   <Display class='Class'>
      <Property name='Name' value='Value'/>
   </Display>
</Field>
```

# Component Classes

HTML components are independent objects that can be combined in various ways. Related components are organized into classes. There are two major groups of component classes:

- **Basic Component classes** -- Components used to display and edit a single value.
- **Container classes --** Components that can contain one or more components.

## Basic Component Classes

Common component classes include the components that are used to display and edit a single value. These components are defined in the section titled *Basic Components*.

## Container Classes

A *container* class defines a collection of components that are visually organized in a certain way. Typically, creating a container class results in the generation of an HTML `table` tag. Simple containers can concatenate the components horizontally or vertically. Other containers allow more flexible positioning of components and may add ornamentation around the components.

Since containers are themselves components, any container can be placed inside another container. You can use this mechanism to build complex page layouts. For example, many pages consist of a title, followed by a list of editing fields, followed by a row of form submission buttons. You can create this by creating a `Panel` component using vertical orientation that contains a `Label` component, an `EditForm` component, and a `ButtonRow` component. The `EditForm` component itself contains some number of subcomponents. The `ButtonRow` component is simply a `Panel` that uses horizontal orientation and contains a list of `Button` components.

### BorderedPanel

Defines five regions (north, south, east, west, and center) into which items can be placed. Components in the north and south regions are positioned horizontally. Components in all other regions are positioned vertically.

Properties include:

- `eastWidth` – Specifies the width of the east region
- `westWidth` – Specifies the width of the west region

## ButtonRow

Sets default options for button placement. Extends the Panel component.

- `pad` – Specifies where to insert this space between the button row and an adjacent component. Allowed values are top and bottom. If the value is null, no space is added. Default value is top.
- `divider` – Specifies whether the divider should be rendered as a horizontal or blank line. When true, the divider will be rendered as a horizontal line (for example, an `<hr>`). (Boolean)

## EditForm

This display component is the default display class used to render forms in a browser.

Form components are positioned in two columns, with titles on the left, and components on the right. Flyover help can be included with the titles. Multiple components can be concatenated on a single row.

Most commonly edited properties include `title`, `subTitle` and `adjacentTitleWidth`.

```
<Form name='Default User Form' help='account/modify-help.xml'>
   <Display class='EditForm'>
     <Property name='titleWidth' value='120'>
     <Property name='adjacentTitleWidth' value='60'>
   </Display>
```

Additional EditForm properties include:

- `adjacentTitleWidth` – Specifies the width of the titles of adjacent fields. If this property is not defined, it defaults to zero. If you define `adjacentTitleWidth` as equal to zero, columns titles will automatically resize. If set to a non-zero value, then the title width of adjacent columns (for example, the second and third columns) will be the value of `adjacentTitleWidth`.
- `componentTableWidth` – Specifies the width (in pixels) of the EditForm. If not specified, this defaults to either 400 pixels or the value of the `defaultComponentTableWidth` global property for EditForm
- `noAlternatingRowColors` – Specifies whether rows in the EditForm are rendered in the same color. When `noAlternatingRowColors` is set to true, every row in the EditForm is rendered the same color. If not specified, this defaults to `false`.

## Panel

The most basic container. Panel renders its children in a simple linear list.

Properties include:

- `horizontal` – Aligns components horizontally, when set to `true`. (Boolean)
- `horizontalPad` – Specifies the number of pixels to use for the cell padding attribute of the table surrounding horizontal components.
- `verticalPad` – Specifies the number of blank lines added between components. (Boolean)

The default orientation is vertical, but can be set horizontal.

## SimpleTable

Arranges components in a grid with an optional row of column titles at the top.

Properties include:

- `columns` – Defines the column headers. Usually a list of message keys, but can also be simple strings. (List)
- `rows` – Defines the cells of the table. Each cell must be a component. (List)
- `columnCount` – Specifies the number of columns if there is no column title list.
- `border` – Determines the width of the table border. Set to 0 to create invisible borders.
- `noItemsMessage` – Specifies the message to display in the table when there are no rows.

## TabPanel

Use to render a tabbed panel that displays a row of tabs as shown below. By default, the tabs are aligned horizontally.

Properties include:

- `leftTabs` – When set to `true`, aligns tabs along left margin, not along the top. (Boolean)
- `border` – Draws a border around the main panel under the tabs, when set to `true`. (Boolean)
- `renderTabsAsSelect` – Renders tabs as a Select drop-down rather than tabs, when set to `true`. This is useful when a form contains many tabs that would cause the browser to scroll horizontally. Do not use in conjunction with aligning the tabs on the left.
- `tabAlignment` – Determines the position of the tabs relative to the page content. Valid values include left (default setting), top, right, bottom, center, and middle.

```
<Field name='MainTabs'>
   <Display class='TabPanel'>
     <Property name='leftTabs' value='false'/>
     <Property name='tabAlignment' value='left'/>
</Field>
```

## Row

Used to create a Panel capable of horizontal alignment.

## SortingTable

Use to create a table whose contents can be sorted by column header.

Properties include:

- `pageButtonAlign` – Determines position of buttons relative to page content. Valid values include left, right, bottom, and center. The default value is right.
- `sortEnable` – Enables column sorting when set to true. (Boolean)
- `sortURL` – Identifies the URL that Identity Manager posts to when column sorting is selected. If column sorting is not set, Identity Manager uses the `_postURL` of the HtmlPage. (String)
- `sortURLParams` – Specifies the parameters that get passed along with the `sortURL`. (String)
- `sortColumn` – Specifies the number of the column that we are currently sorting by. The default is to set this value to the first column. (Integer)
- `sortOrder` – Specifies the sort order. Values includes `asc` (for ascending) or `desc` (for descending). Default value is `asc`. (String)
- `linkEnable` – Indicates if this table is to be generated with the first column as links. (Boolean)
- `linkURL` – Specifies the URL that Identity Manager links to when generating links. If not specified, defaults to the post URL of the containing HtmlPage. (String)
- `linkURLArguments` – Indicates the arguments to include in the link URL.
- `linkColumn` – Specifies the column number that will be used for the generated links as specified by the `linkURL` attribute. (Integer)
- `linkParameter` – Specifies the name of the post data parameter that will have the value of the link row `id`. The default value is `id`.
- `selectEnable` – Indicates whether a column of checkboxes is displayed along a multiselect table's left margin. When set to `true`, Identity Manager displays a column of checkboxes. (Boolean)
- `columns` – Lists table column headers. (List of strings)

- `pageSize` – Specifies that the table should display at most `_pageSize` entries simultaneously. If more than `_pageSize` entries exist, then interface elements allow paging through the results. If `_pageSize` is less than 1 (the default setting), then all entries are displayed at once. (Integer)
- `useSavedPage` – If the value of `pageSize` exceeds 0, then the sorting table saves the current sorting table page on the HTTP session in the `<fieldName>_currentPage` attribute. The `_useSavedPage` property indicates whether the current page should be retrieved from the HTTP session and displayed. By making the value of this property the result of an XPRESS expression, the form or view can control when the current page is recalled after when returning back to the JSP containing the `SortingTable` component. (Boolean)

  For example, if the `SortingTable` component displays the results of a query containing editable items, to ensure that Identity Manager displays the results page that contains the edited item after the user has edited an item in the result table, enter a value that exceeds 0.

## WizardPanel

Use to render one of several child components (typically EditForms) that use wizard-style Next and Previous buttons to navigate between components.

Properties include:

- `button` – Specifies a value for child component's `location` property that will place it in the button row. (String)
- `nextLabel` – Specifies the label to display on the Next button. The default text is Next. (String)
- `prevLabel` – Specifies that the label in the Previous button is displayed. (String)
- `cancelLabel` – Specifies that the label in the Cancel button is displayed. (String)
- `okLabel` – Specifies that the label is displayed in the OK button. (String)
- `noOk` – Specifies that the OK button is not displayed. (Boolean)
- `alwaysOk` – Determines that the OK button is displayed, when set to `true`. (Boolean)
- `noCancel` – Specifies that the Cancel button is not displayed, when set to `true`. (Boolean)
- `topButtons` – Causes the buttons to be rendered at the top of the page rather than the page bottom, when set to `true`. (Boolean)
- `noButtons` – Suppresses all button rendering when set to `true`. (Boolean)

## Page Processor Requirements for HTML Components

This section describes the page processor requirements to display forms that implement HTML components.

### Hidden Parameters

Most components have a name that corresponds to the name of a parameter posted from an HTML form. Identity Manager reserves a few parameter names for general use. Do not use these names as component names.

| Reserved Name | Description |
|---|---|
| `id` | Contains the ID of the object being edited |
| `command` | Contains the value of the button used to submit the form |
| `activeControl` | Contains the name of the last component that was active on the form |
| `message` | Can contain an informational message to be displayed at the top of the page |
| `error` | Can contain an error message to be displayed at the top of the page |

# Component Subclasses

All components extend the `Component` class, which defines the properties common to most components. In addition, some components extend the `Container` class, which gives them the ability to contain other components.

Each `Component` subclass defines a number of *properties* that are used to specify the characteristics of the component beyond those implied by the Component base class. For example, the `Label` component supports a `font` property, which can be used to specify the font used when rendering the label.

## Naming Conventions

Properties always begin with a lowercase letter and use camel case to separate adjacent words. Access method names are formed by capitalizing the property name, and prefixing either `get` or `set`. For example, the property named `font` is accessible from Java using the following methods:

- `getFont`
- `setFont`

The data type for each property varies and is documented with the property. the terminology used to describe property value types is described in the following table.

## Data Types

This table lists the data types allowed in component properties.

| Type | Description |
|---|---|
| `null` | Indicates that a property has no value |
| `String` | Represents the most common data type. String values are usually represented by an instance of the Java `String` class. Some components are values of any class. These are implicitly coerced to strings with the `toString` method.<br>Unless otherwise specified, you can assume that all properties are of type `string`.<br>**Example** `<String>Hello World</String>` |
| `List of string` | Indicates that the value is expected to be a list of one or more strings. In Java, this value is always implemented as an instance of the `List` class. The elements of the list are then expected to be instances of the `String` class.<br>**Example**<br>`<List>`<br>`  <String>choice one</String>`<br>`  <String>choice two</String>`<br>`</List>` |

| Type | Description |
|---|---|
| `comma list` | Represents a single string that contains character ranges separated by a comma character. Components that have multivalued properties typically allow either `list of string` values or `comma list` values. When a comma list is assigned to a property, the component converts it to a list of strings by extracting the substrings between commas and adding them to a list in the order in which they appear in the string. |
| `boolean` | Represents a logically `true` or `false value`. Many properties require Boolean values.<br>A Boolean value can be represented several ways. `false` represents one of the following values:<br>• a null value<br>• a Boolean object whose value is false<br>• an Integer object whose value is 0<br>• A string whose value is anything other than true.<br>When defining components in XML, Boolean values are most often represented with the strings `true` and `false`. (Any string value other than `true` is considered false.) |
| `int` | Represents a positive or negative integer. `int` properties are commonly specified using strings. In that case, the string is coerced to an integer using the usual rules. If a string contains invalid integer characters, the coerced value is zero. |

## Base Component Class

The `Component` class is the base class for all HTML components. It contains the properties that are common to most components. Not all `Component` properties are relevant in every subclass. For example, `Component` defines a property `allowedValues` that can contain a list of value constraints. This property is relevant only in subclasses that allow value editing such as `Select` or `MultiSelect`. Further, `Container` classes almost never directly represent an editable value. Consequently, any properties related to the component value are irrelevant. Some properties are relevant only if the component is contained within a specific `Container` class.

### name

Specifies the internal name of a field. All editing components must have a name, which is typically unique among all components displayed on the page. `name` is a string that is usually a path to a view attribute.

Container components do not require names and any assigned names are ignored. When building components from Java, component names are defined by the application. When building components from XML forms, component names are derived from the names of `Field` elements in the form. Field names are in turn *path expressions* within the view object that is used with the form.

#### Example

```
<Field name ='global.firstname'>
```

For more information on how the `name` attribute refers to a specific attribute in the user view, see *Views*.

### title

(Optional) Specifies the external name of a field. Titles are typically used with the `EditForm` container, which builds an HTML table that contains titles in one column and components in another.

Components do not render their own titles. Rendering of titles is controlled by the container. Many containers ignore titles.

#### Examples

```
<Property name='title' value='FirstName'/>
<Property name='title'>
    <expression>
       <concat>
           <s>Edit User: </s>
          <ref>waveset.accountId</ref>
      </concat>
   </expression>
</Property>
```

In this example, the field title is in part derived dynamically from the user's Identity Manager account ID.

## value

Editing components have a value that may be null. The value is typically set automatically by Identity Manager from an attribute in a view. Some components allow you to set the value by explicitly ignoring current view content. This value can be null.

The `Component` class allows the value to be any Java object. The subclass must coerce the value to a particular type when it is assigned, or when the HTML is generated. Component values are almost always `String` objects or `List` objects that contain strings. See the section titled *Data Types* for more information on component value types.

Most container classes do not have values. If you assign a value, it is ignored. Some containers do allow values (for example, `TabPanel` and `WizardPanel`).

When building components from XML forms, the value is usually derived by using the component `name` as a path into the underlying view object, which contains all the values being edited.

### Example

```
<Property name= 'value' value='false '/>
```

## allowedValues

Specifies an optional list of allowed values for the component. If specified, the component allows you to select from only values that are on the list. If the component supports value restrictions, the list of allowed values is stored here. The value is always a list and usually contains strings. For convenience when setting properties from XML forms, you can also specify the allowed values as a comma list.

For more information on using this property to populate lists, see the section titled *Lists*.

### Example

```
<Property name='allowedValues' value= 'Mon, Tue, Wed, Thurs,
 Fri'/>
<Property name='allowedValues'/>
   <expression>
      <call name='DaysoftheWeek'/>
   </expression>
</Property>
```

## primaryKey

This property is recognized only by the `SortingTable` container. The `SortingTable` container organizes components into a table with each column expected to contain components of the same class. `SortingTable` allows the rows to be sorted according to the values in any column. Typically, the sort order is determined from the `value` of each component in the column. There may be cases, however, where the value of the component is not suitable for sorting or may be inefficient to compare. In these cases, you can specify an alternate numeric sorting key.

## required

If `true`, indicates that the field is expected to have a value before the form is submitted. If the component is contained within an `EditForm`, a red * (asterisk) will be placed after the component to indicate that the user must enter a value before saving. If the required schema map attribute is selected, (that is, set to a value of `true)`, the field is always required.

The value of the property must be either `true` or `false`.

### Example

```
<Property name='required ' value='true '/>
```

## noNewRow

If `true`, the field displays on the Identity Manager page next to the previous field. If not specified or set to `false`, the field appears on a new line, directly under the previous field. The default value is `false`

This Boolean property is recognized only if the field is contained in a form that uses the `EditForm` display class. Typically, `EditForm` renders each component on a new row with the titles aligned in the left column and the component in the right column. To conserve space, you can concatenate several components on the same row. If the component also has a title, the title is rendered as non-highlighted text between the previous component and this component.

Values include:

```
value='true ' | 'false '
```

### Example

```
<Property name='noNewRow ' value='true '/>
```

### location

Use if the container defines more than one display area and the component must be added to a specific area. Some containers allow the placement of components to be controlled by assigning a value to the `location` property. For example, the `BorderedPanel` container supports five different display areas: north, south, east, west, and center.

The recognized values for the `location` property are defined by the container. If you do not assign a location, or assign a location name that is not recognized, the container places the component in the default location.

### help

Specifies text that may be displayed to assist the user in understanding purpose of the field. In most Identity Manager pages, this will cause the `<icon>` icon to be displayed next to the component title. Moving the mouse over this icon will cause the help text to be displayed in the left margin.

The value of the property can either be literal text to be displayed, or it can be a message catalog key. Literal text can include HTML markup.

For more information on adding help to your custom form, see *Adding Guidance Help to Your Form* in the chapter titled *Forms.*

### inlineHelp

Specifies the text that can be rendered beneath a component in Identity Manager pages.

The value of the property can either be literal text to be displayed, or it can be a message catalog key. Literal text can include HTML markup.

## command

Specifies a command to submit when a component is modified. (When a user makes a change to a value, form output is recalculated.)

This property is typically used with the `Button` component. Some components must cause immediate submission of the surrounding HTML form when they are modified so that the application can regenerate the page based on that modification. Setting the `command` property to a non-null value causes this behavior.

When the `command` property is set, and the component is modified, the form is posted and an extra hidden parameter named `command` is posted whose value is the value of the `command` property.

The command specifies how the system will process the edits that have been made to a view. The `command` property must have one of the following values.

| Value | Description |
|---|---|
| **Save** | Causes the edits to be saved. |
| **Cancel** | Causes the edits to be discarded. |
| **Recalculate** | Causes the page to be regenerated. |
| **SaveNoValidate** | Causes the edits to be saved, but no form validation to be performed. |

Since specifying a command value of `Recalculate` is so common in forms, an shorter alterative syntax is available. The `Display` element has an attribute named `action` that when set to true, has the same effect as setting the `command` property to **Recalculate**.

```
<Display class='Select' action='true'>"
```

## onClick

When specified, the value is expected to contain JavaScript that will be assigned as the value of the `onClick` attribute of the `input` element generated for this component. Not all components support the `onClick` property.

Use of this property is rare and requires detailed knowledge of the generated HTML. If you use this property, the page must typically contain a `Javascript` component that defines JavaScript functions you call from within the `onClick` value.

**Example**

```
<Property name='onClick' value="Uncheck(this.form,
'resourceAccounts.selectAll');"/>
```

**Note** Once forms are stored in the repository, Identity Manager always uses single quotes to surround attribute values. If single quotes appear within the attribute value, they will be replaced with '. To prevent this escaping you can represent the string in an XPRESS `s` expression:

```
<Property name='onClick'>
   <s>Uncheck(this.form, 'resourceAccounts.selectAll'); </s>
</Property>"
```

## onChange

Similar to `command`. The value can be an arbitrary JavaScript statement to run when the field is modified.

Not all components support the `onChange` property.

Use of this property is rare and requires detailed knowledge of the generated HTML. If you use this property, the page must typically contain a `Javascript` component that defines JavaScript functions you call from within the `onChange` value.

## nowrap, align, width, valign, and colspan

Most containers position subcomponents by surrounding them with an HTML `table` tag. The HTML generated for each component then is typically contained in a `td` tag. Some containers can recognize the `nowrap`, `align`, `width`, and `colspan` properties and use them when generating the surrounding table cell tag. You can use these components to adjust the position and size of the component within the container.

- `nowrap` – Specifies how some components are displayed if they contain a long string of text. If the value of `nowrap` is false or unspecified, the browser can break up the component text into multiple lines when it is displayed. If the value of `noWrap` is true, the browser will try to keep the component text on a single line.
- `align` – Rarely used. Adjusts the element horizontally on the form. Allowed values are left, right, and center.
- `valign` – Rarely used. Specifies where components are rendered vertically. Allowed values are top, bottom, and middle.
- `colspan` – Deprecated

### Examples

```
<Property name= 'width' value='3'/>
<Field name='Start Day' prompt='Day' nowrap='true'/>
```

# Basic Components

## BackLink

Displays a link that returns to the previous page. The behavior of this component is the same as that of the browser **Back** button. However, you may want to place this link in a convenient position on the page.

Properties for this display component:

- `text` – Specifies the text of the link. If you do not specify text, the link defaults to **Back**.

### Example

```
<Field name='back'>
   <Display class='BackLink'>
      <Property name='value' value='previous page'/>
   </Display>
</Field>
```

## Button

Displays a button. Buttons typically submit the surrounding form, but they can also be defined to run arbitrary JavaScript.

Properties for this display component are:

- `name` – Specifies the name of the parameter that will be posted when the user clicks this button. This property is optional; if not specified, the default value is `command`.
- `value` – Specifies the value of the parameter posted when the user clicks this button.
- `label` – Specifies the visible text that displays on the button.
- `command` – Specifies an optional value to submit along with the `name` parameter (for example, Save, Cancel, Recalculate).
- `postURL` – Specifies an alternate, target URL to which the form will be posted. This value overrides the URL specified in the JSP.
- `hiddenID` – Specifies an optional value for an *id* parameter to be included in the form post data.

**Example**

```
<Display class='Button'>
   <Property name='label' value='Change Password'/>
   <Property name='value' value='Recalculate'/>
</Display>
```

## Checkbox

Displays a checkbox. When checked, the box represents a value of `true`. An unselected box represents a `false` value.

Properties for this display component are:

- `label` – (Optional) Specifies a label that is displayed to the right of the checkbox. It is displayed adjacent to the component, but is not displayed in the title column
- `leftLabel` – Specifies that the label should appear to the left of the checkbox.
- `checkAll` – Set when this checkbox is serving as a select all checkbox, which should then propagate its value to a set of other checkboxes. The value of the property is a regular expression that is used to match the names of other checkboxes on the HTML page.
- `uncheck` – Set to the name of another checkbox field that represents the select all checkbox in a collection of synchronized checkboxes. If this is set, whenever the check status of this checkbox is changed, the select all checkbox is unchecked.
- `syncCheck` – Set to the name of another checkbox field that must stay in sync with the value of the checkbox field on which this property is set. If this is set, whenever the value of this checkbox is changed, the sync'ed checkbox's value is set to the same value.
- `syncUncheck` – Set to the name of another checkbox field that should stay in sync when the value of the checkbox field on which this property is set is changed to `false`. If this is set, whenever the value of this checkbox is changed to `false`, the sync'ed checkbox's value will also be set to `false` (unchecked).
- `syncCheckAllTo` – Indicates that all select-all checkboxes matching the regular expression will be kept in sync with the value of the checkbox field on which this property is set when its value is changed to `false`. The value of this property is a regular expression that represents one or more of the select-all checkboxes.

- `syncUncheckAll` – Set to the name of another checkbox field that should stay in sync when the value of the checkbox field on which this property is set is changed to `false`. If this is set, whenever the value of this checkbox is changed to `false`, the sync'ed checkbox's value will also be set to `false` (unchecked).
- `syncCheckTo` – Indicates that all checkboxes matching the regular expression will be kept in sync with the value of the checkbox field on which this property is set. Whenever the value of the checkbox field on which this property is set is changed, the sync'ed checkbox's value will be set to the same value. The value of the property is a regular expression.
- `value` – Determines the state of the checkbox. If the value is logically true, the checkmark appears.

**Example**

```
<Field name='accounts[NT].passwordExpired'>
   <Display class='Checkbox'>
      <Property name='title 'value='Password is Expired'/>
   </Display>
</Field>
```

## DatePicker

Allows the user to specify a date using an applet that displays a calendar. The field is displayed in the form as a calendar icon. When the icon is clicked, the calendar applet is launched in a separate window.

Properties include:

- `format` – Specifies the date format to use for displaying the date. This can be a Java-style date formatting string that uses any of the following formatting characters: `y`, `M`, or `d`. This can also be the value `iso`, specifying ISO format (yyyy-MM-dd), or the value `local`, specifying a locale-sensitive format (the Java default for the locale). If omitted, the format "MM/dd/yyyy" will be used.
- `multiField` – Indicates whether separate input fields should be displayed for each element of the date. If omitted or false, a single text field will be used for input, expecting properly formatted date text.
- `value` – Specifies the date to be highlighted on the calendar as the current date. Date can be parsed from either a `Date` object or a `String` object.

**Example**

```
<Field name='ExpireDate'>
   <Display class='DatePicker'>
      <Property name='title' value='Set Password Expire  date'/>
      <Property name='format'= value='iso'/>
   </Display>
</Field>
```

## FileUpload

Displays a text field and a **Browse** button that allows the user to select a file and upload it to the server. Use this component to import data into Identity Manager from a file (such as users or configuration objects). This component supports all the properties that the `Text` component supports.

## Html

Describes the root HTML page. This component can contain arbitrary HTML and browser JavaScript. Properties include:

- `commentScripts` – Indicates whether `<script>` tags emitted for JavaScript should be enclosed in comments.
- `title` – Specifies the title of the page. Can be a String or Message, but typically is a String.
- `postUrl` – Specifies the URL that Identity Manager posts to when the main form is submitted.
- `messages` – Indicates which informational messages to display.
- `comments` – Indicates the special comments to include. This property is typically used by GenericEditForm and FormConverter when these methods catch exceptions.
- `foccussedFieldName` – Specifies the name of the first field to receive focus. Typically null. The value of this property is calculated as the first text field, or if no text fields, the first field.
- `activeControl` – Specifies the name of the last known active form field. (String)

## Javascript

Use to insert pre-formatted JavaScript into the page. This is useful if you are using the `onClick` or `onChange` properties in components and want to call custom JavaScript functions.

Though not required, it is recommended that you specify the `name` property when building components from XML forms. Using features such as field loops and field inclusion, you can add more than one JavaScript component containing the same script to the page. During HTML generation, JavaScript components that have the same name are included only once.

The component has an extended property named `script` that can contain the JavaScript text.

You can also include JavaScript by setting the property named `source`. This should be a string containing a URL fragment relative to the base context. It is the JavaScript contained in the indicated file to be loaded by the browser.

## Label

Displays a string of text.

Properties for this display component are:

- `value` – Defines the text to be displayed. The value can be either a string or a list of strings. When the value is a list, each string in the list is displayed on a separate line.
- `leftPad` – Specifies the number of spaces to insert to the left of the label.
- `pad` – Specifies the number of spaces to insert to the left and right of the label.
- `rightPad` – Specifies the number of spaces to insert to the right of the label.

**Note** If no padding is specified, the default padding is leftPad=2, rightPad=2.

### Example

```
<Field>
   <Display class='Label'>
      <Property name='title' value='Account ID'/>
      <Property name='value'>
         <ref>waveset.accountId,/ref>
      </Property>
   </Display>
</Field>
```

- `font` – Specifies the font style. The value must be one of the styles defined in the `styles/style.css` file of the Identity Manager installation directory.
- `color` – Specifies the label color. Use standard HTML color formatting (`#`*xxxxxx*) to specify the color value.

## Link

Places a link on the page.

Properties include:

- `URL` – Specifies the target Uniform Resource Locator (URL).
- `imageURL` – (Optional) Specifies the URL to an icon or image that will be rendered to the right of the link.
- `imageURL2` – (Optional) Specifies the URL to an icon or image used will be rendered to the right of the first image.
- `hoverText` – Specifies text to display when the mouse rests over the first or second image.
- `id` – (Optional) Specifies a value to be included as the *id* query argument in the link.
- `arguments` – (Optional) Specifies a set of name/value pairs to be included as query arguments.
- `extraURL` – (Optional) Specifies an additional URL fragment to be included after the base URL and arguments.

**Example**

```
<Field>
   <Display class='Link'>
      <Property name='name' value='Request
            Group Access'/>
      <Property name='URL'
         value='user/processLaunch.jsp?newView=true'>
      <Property name='id' value='Group Request
            Process'/>
   </Display>
</Field>
```

**Note** Link components are one place in your form where you might use a `<map>` element to pass name/value pairs. In the following example, the `<map>` element contains several pairs: a mapping of a String to a Boolean value and a String to a List.

```
<invoke class='com.waveset.ui.FormUtil'
name='getOrganizationsDisplayNames'>
   <ref>:display.session</ref>
     <map>
       <s>filterVirtual</s>
       <o><Boolean>true</Boolean></o>
       <s>current</s>
       <list>
         <ref>original.orgParentName</ref>
```

```
        </list>
        <s>excluded</s>
        <list><ref>orgName</ref></list>
      </map>
</invoke>
```

## LinkForm

Renders a bulleted list of links, resembling a menu.

## NameValueTable

A component that renders a collection of name/value pairs in a simple two column table. This component directly renders the data it contains.

Data can be specified in several forms:

- flat list – The list is expected to contain name/value pairs such that element 0 is a name, element 1 is a value, element 2 is a name.
- map – The entries in the map are emitted in alphabetical order.
- GenericObject – The object is flattened to and emitted as a map.

Properties include `_hideEmptyRows`, which when set to `true`, hides rows for which no value exists.

## MultiSelect

Displays a multiselection text box, which displays as a two-part object in which a defined set of values in one box can be moved to a selected box. Values in the left box are defined by the `allowedValues` property, values are often obtained dynamically by calling a Java method such as `FormUtil.getResources`. The values displayed in the right side of a multiselection box are populated from the current value of the associated view attribute, which is identified through the field name.

The form titles for this two-part object are set through the `availabletitle` and `selectedtitle` properties.

If you want a MultiSelect that does not use an applet, set the `noApplet` property to true.

**Note** If you are running Identity Manager on a system running the Safari browser, you must customize all forms containing MultiSelect components to set the `noApplet` option. Set this option as follows:

```
<Display class='MultiSelect'>
    <Property name='noApplet' value='true'/>
...
```

Properties for this display component are:

- `availableTitle` – Specifies the title of the available box.
- `selectedTitle` – Specifies the title of the selected box.
- `ordered` – Defines whether selected items can be moved up or down within the list of items in the text box. A `true` value indicates that additional buttons will be rendered to permit selected items to be moved up or down.
- `allowedValues` – Specifies the values associated with the left side of the multiselection box. This value must be a list of strings. **Note**: The `<Constraints>` element can be used to populate this box, but its use is deprecated.
- `sorted` – Specifies that the values in both boxes will be sorted alphabetically.
- `noApplet` – Specifies whether the multiselect will be implemented with an applet or with a pair of standard HTML select boxes. The default is to use an applet, which is better able to handle long lists of values. See preceding note for information on using this option on systems running the Safari browser.
- `typeSelectThreshold` – (Available only when the `noApplet` property is set to true.) Controls whether a type-ahead select box appears under the `allowedValue` list. When the number of entries in the left select box reaches the threshold defined by this property, an additional text entry field appears under the select box. As you type characters into this text field, the select box will scroll to display the matching entry if one exists. For example, if you enter **w**, the select box scrolls to the first entry that begins with **w**.
- `width` – Specifies the width of the selected box in pixels. The default value is 150.
- `height` – Specifies the width of the selected box in pixels. The default value is 400.

### Example

```
<Field name='accounts[LDAP].LDAPDept' type='string'>
   <Display class='MultiSelect' action='true'>
      <Property name='title' value='LDAP Department'/>
   </Display>
   <Constraints>
      <o>
         <List>
             <String>Sales</String>
             <String>Marketing</String>
             <String>International Sales</String>
      </List>
      </o>
   </Constraints>
</Field>
```

## Radio

Displays a horizontal list of one or more radio buttons. A user can select only one radio button at a time. If the component value is null or does not match any of the allowed values, no button is selected.

Properties for this display component are:

- `title` – Specifies the title for all radio buttons.
- `labels` – Specifies an alternate list of button labels. The `labels` list must be as long as the values in the `allowedValues` list. Alternate labels can be used in cases where the values are cryptic. For example, values can be letter codes such as H, M, and S, but you would use this property to identify button labels hours, minutes, and seconds.
- `allowedValues` – Specifies the value associated with each button. This value must be a list of strings.
- `value` – Specifies values for the buttons. This value accepts one string. If not set, then the values are the same as the labels.

### Example

```
<Field name='attributes.accountLockExpiry.unit'>
   <Display class='Radio'>
     <Property name='noNewRow' value='true'/>
     <Property name='labels'>
        <List>
           <String>UI_TASKS_XML_SCHED_MINUTES</String>
           <String>UI_TASKS_XML_SCHED_HOURS</String>
           <String>UI_TASKS_XML_SCHED_DAYS</String>
           <String>UI_TASKS_XML_SCHED_WEEKS</String>
           <String>UI_TASKS_XML_SCHED_MONTHS</String>
        </List>
```

```
    </Property>
    <Property name='allowedValues'>
       <List>
          <String>minutes</String>
          <String>hours</String>
          <String>days</String>
          <String>weeks</String>
          <String>months</String>
       </List>
    </Property>
  </Display>
</Field>
```

## SectionHead

Displays a new section heading defined by the value of the `text` property. It is an extension of the `Label` class that sets the `font` property to a style that results in large bold text. It also sets the `pad` property to zero to eliminate the default 2 space padding. Use it to break up long forms into sections separated by a prominent label.

The only property for this display component is `text`, which specifies the text to be displayed.

### Example

```
<Field>
   <Display class='SectionHead'>
      <Property name='text' value ='Calculated Fields'/>
   </Display>
</Field>
```

## Select

Displays a single-selection list box. Values for the list box must be supplied by the `allowedValues` property.

Properties for this display component are:

- `allowedValues` – Specifies the list of selectable values for display in the list box.
- `allowedOthers` – When set, specifies that initial values that were not on the `allowedValues` list should be tolerated and silently added to the list.
- `autoSelect` – When set to `true`, this property causes the first value in the `allowedValues` list to be automatically selected if the initial value for the field is null.
- `multiple` – When set to `true`, allows more than one value to be selected.
- `nullLabel` – Specifies the text that displays at the top of the list box when no value is selected.

- `optionGroupMap` – Allows the selector to render options in groups using the `<optgroup>` tag. Format the map such that the keys of the maps are the group labels, and the elements are lists of items to be selectable. (Values must be members of `allowedValues` in order to render.)
- `size` – (Optional) Specifies the maximum number of rows to display. If the number of rows exceeds this size, a scroll bar is added.
- `sorted` – When set to `true`, causes the values in the list to be sorted.
- `valueMap` – Maps raw values to displayed values.

The component supports the `command` and `onChange` properties.

**Example**

```
<Field name='city' type='string'>
   <Display class='Select'>
      <Property name='title' value='City'/>
      <Property name='allowedValues'>
         <List>
            <String>Austin</String>
            <String>Portland</String>
            <String>New York</String>
         </List>
      </Property>
   </Display>
</Field>
```

## Text

Displays a regular text entry box.

Common properties for this display component are:

- `size` – Specifies the number of characters that are visible in the text entry box. The box size is recalculated depending upon the length of the text in the box.
- `notrim` – Specifies whether text posted from the HTML form is trimmed. Set to `true` to not trim white space. To preserve white space, set this option.
- `maxLength` – Specifies the maximum length of the string that can be edited in the text box.
- `multiValued` – Displays text boxes with Add and Remove buttons to add and remove values, when set to `true`.
- `secret` – Displays ***** (asterisks) in the place of entered text. This option is most often used in password fields.

- `readOnly` – Displays read-only text. This text cannot be edited by the user. You might use this property if, for example, you want to display resource attribute information that an administrator needs to view when creating or editing user accounts.
- `submitOnEnter` – When this property is set and the `Text` field has focus, then when the user presses the Enter key, the form is submitted using the command that is specified in the property value. In the following example, the form is submitted exactly as though the user has clicked **Save**.

**Example**

```
<Field name='variables.identityID'>
   <Display class='Text'>
      <Property name='required'>
         <Boolean>true</Boolean>
      </Property>
      <Property name='title' value='Identity ID'/>
      <Property name='size' value='32'/>
      <Property name='maxLength' value='128'/>
      <Property name='submitOnEnter' value='Save'/>
   </Display>
</Field>
```

## TextArea

Displays a multi-line text entry box.

Properties for this display component are:

- `rows` – Specifies the number of text area rows. (Integer)
- `columns` – Specifies the number of text area columns. (Integer)
- `readOnly` – Displays read-only text in the text entry box. When set to `true`, this component will not have a border. (Boolean)
- `format` – Set to control how `setValue()` behaves and determine the type of object returned by `getPostData()`. (String)
- `sorted` – Enables sorting of lines in the text area, when set to `true`. This feature is convenient when the area is used to display a list of selections, not free-form text. (Boolean)
- `noTrim` – Specifies whether text posted from the HTML form is trimmed. The default is to trim white space. To preserve white space, set this value to `true`.

**Example**

To display a text box with five visible rows that wraps after each 70 characters specify:

```
<Field name='Description'>
   <Display class='TextArea'>
     <Property name='rows' value='5'/>
     <Property name='columns' value='70'/>
   </Display>
</Field>
```

If the user enters text beyond the defined visible rows, the text area displays a scroll bar.

# A Form and Process Mappings

This appendix lists the forms and workflow processes used in Identity Manager and their corresponding system names.

The **Form Mappings** column lists the system name of the form.

The name listed in the **Form Name Mapped To** column is the name by which the form is identified in the Business Process Editor (BPE) and in the Debug page of Identity Manager.

| Form Mappings | Form Name Mapped to |
|---|---|
| `LDAP ChangeLog ActiveSync Create Group Form` | LDAP Create Group Form |
| `LDAP ChangeLog ActiveSync Create Organization Form` | LDAP Create Organization Form |
| `LDAP ChangeLog ActiveSync Create Organizational Unit Form` | LDAP Create Organizational Unit Form |
| `LDAP ChangeLog ActiveSync Create Person Form` | LDAP Create Person Form |
| `LDAP ChangeLog ActiveSync Update Group Form` | LDAP Update Group Form |
| `LDAP ChangeLog ActiveSync Update Organization Form` | LDAP Update Organization Form |
| `LDAP ChangeLog ActiveSync Update Organizational Unit Form` | LDAP Update Organizational Unit Form |
| `LDAP ChangeLog ActiveSync Update Person Form` | LDAP Update Person Form |
| `LDAP Listener ActiveSync Create Group Form` | LDAP Create Group Form |
| `LDAP Listener ActiveSync Create Organization Form` | LDAP Create Organization Form |
| `LDAP Listener ActiveSync Create Organizational Unit Form` | LDAP Create Organizational Unit Form |
| `LDAP Listener ActiveSync Create Person Form` | LDAP Create Person Form |

| | |
|---|---|
| LDAP Listener ActiveSync Update Group Form | LDAP Update Group Form |
| LDAP Listener ActiveSync Update Organization Form | LDAP Update Organization Form |
| LDAP Listener ActiveSync Update Organizational Unit Form | LDAP Update Organizational Unit Form |
| LDAP Listener ActiveSync Update Person Form | LDAP Update Person Form |
| Windows Active Directory ActiveSync Create Container Form | Windows Active Directory Create Container Form |
| Windows Active Directory ActiveSync Create Group Form | Windows Active Directory Create Group Form |
| Windows Active Directory ActiveSync Create Organizational Unit Form | Windows Active Directory Create Organizational Unit Form |
| Windows Active Directory ActiveSync Create User Form | Windows Active Directory Create User Form |
| Windows Active Directory ActiveSync Update Container Form | Windows Active Directory Update Container Form |
| Windows Active Directory ActiveSync Update Group Form | Windows Active Directory Update Group Form |
| Windows Active Directory ActiveSync Update Organizational Unit Form | Windows Active Directory Update Organizational Unit Form |
| Windows Active Directory ActiveSync Update User Form | Windows Active Directory Update User Form |
| accountOwnerSelection | Account Owner Selection Form |
| anonymousUserMenu | Anonymous User Menu |
| changeAnswers | Change User Answers Form |
| changeCapabilities | Change User Capabilities Form |
| changeMyPassword | Change My Password Form |
| changePassword | Change User Password Form |
| changePasswordSelection | User Selection Form |
| confirmDeletes | Confirm Deletes |

| | |
|---|---|
| deprovisionUser | Deprovision Form |
| disableUser | Disable Form |
| editArgument | Edit Argument |
| editField | Edit Field |
| editForm | Edit Form |
| editRule | Edit Rule |
| enableUser | Enable Form |
| endUserChangePassword | Change Password Form |
| endUserForm | End User Form |
| endUserLaunchList | End User Launch List |
| endUserMenu | End User Menu |
| endUserResetPassword | Reset User Password Form |
| endUserTaskList | End User Task List |
| endUserTaskResults | End User Task Results |
| endUserWorkItemEdit | End User Work Item Edit |
| endUserWorkItemList | End User Work Item List |
| findAccountOwner | Find Account Owner Form |
| findUser | Find User Form |
| findUserResults | Find User Results Form |
| listForms | List Forms |
| listRules | List Rules |
| loadForm | Default User Form |
| loginChangePassword | Expired Login Form |
| loginResetPassword | Reset User Password Form |
| renameUser | Rename User Form |
| reprovisionForm | Default User Form |
| resetPassword | Reset User Password Form |
| resetPasswordSelection | User Selection Form |

| | |
|---|---|
| selfDiscovery | Self Discovery |
| userForm | Tabbed User Form |
| viewUserForm | Tabbed View User Form |

The **Process Type** column lists the system name of the form.

The name listed in the **Process Name Mapped To** column is the name by which the process is identified in the Business Process Editor (BPE) and in the Debug page of Identity Manager.

| Process Type | Process Name Mapped to... |
|---|---|
| changeResourceAccountPassword | Change Resource Account Password |
| changeUserPassword | Change User Password |
| createResourceGroup | Create Resource Group |
| createResourceObject | Create Resource Object |
| createResourceOrganization | Create Resource Organization |
| createResourceOrganizationalUnit | Create Resource Organizational Unit |
| createResourcePerson | Create Resource Person |
| createResourceUser | Create Resource User |
| createUser | Create User |
| deleteAccount | Delete Resource Account |
| deleteResourceGroup | Delete Resource Group |
| deleteResourceObject | Delete Resource Object |
| deleteResourceOrganization | Delete Resource Organization |
| deleteResourceOrganizationalUnit | Delete Resource Organizational Unit |
| deleteResourcePerson | Delete Resource Person |
| deleteResourceUser | Delete Resource User |
| deleteUser | Delete User |

| | |
|---|---|
| disableUser | Disable User |
| enableUser | Enable User |
| handleNativeChangeTo AccountAttributes | Audit Native Change To Account Attributes |
| lockUser | Lock User |
| manageResource | Manage Resource |
| manageRole | Manage Role |
| questionLogin | Question Login |
| renameUser | Rename User |
| resetUserPassword | Reset User Password |
| unlinkResourceAccountsFrom User | Unlink Resource Accounts From User |
| unlockUser | Unlock User |
| updateResourceGroup | Update Resource Group |
| updateResourceObject | Update Resource Object |
| updateResourceOrganization | Update Resource Organization |
| updateResourceOrganization alUnit | Update Resource Organizational Unit |
| updateResourcePerson | Update Resource Person |
| updateResourceUser | Update Resource User |
| updateUser | Update User |

# Index

## C



## D

## E



## F

## G



## H



## I



## J

## T



## U



## V

## W

## X