SailPoint IdentityIQ

Version 8.1

System Administration Guide

Table of Contents

IdentityIQ Introduction

SailPoint IdentityIQ is an identity and access management solution for enterprise customers that delivers a wide variety of IAM processes-including automated access certifications, policy management, access request and provisioning, password management, and identity intelligence. Furthermore, IdentityIQ has a flexible connectivity model that simplifies the management of applications running in the datacenter or the cloud.

Compliance Manager — IdentityIQ Compliance Manager automates access certifications, policy management, and audit reporting through a unified governance framework. This enables you to streamline compliance processes and improve the effectiveness of identity governance, all while lowering costs.

Lifecycle Manager — IdentityIQ Lifecycle Manager manages changes to access through user-friendly self-service request and password management interfaces and automated lifecycle events. It provides a flexible, scalable provisioning solution for addressing the constantly evolving access needs of your business in a way that's both efficient and compliant.

IdentityAI – Integrating IdentityAI within IdentityIQ enables the delivery of Predictive Identity. IdentityAI is a rule based machine learning engine using identity graph technology to provide recommendations for access review and access request decisions. With IdentityAI enabled, you can also review access history for identity cubes, create dashboards that can be customized from an administrative perspective, and view peer groups within the IdentityAI user interface.

Privileged Account Management Module — IdentityIQ Privileged Account Management module provides a standardized approach for extending critical identity governance processes and controls to highly privileged accounts, enabling IdentityIQ to be used as a central platform to govern standard and privileged accounts.

Connectors and Integration Modules — IdentityIQ offers Integration Modules that support the extended enterprise IT infrastructure. Third party provisioning and service desk integration enable multiple sources of fulfillment to access change. Service catalog integration supports a unified service request experience with integrated governance and fulfillment. Mobile device management integration mitigates risk posed by mobile devices through centralized visibility, control and automation. And IdentityIQ's IT security integration provides enhanced security with improved responsiveness and controls.

Open Identity Platform — SailPoint's Open Identity Platform lays the foundation for effective and scalable IAM within the enterprise. It establishes a common framework that centralizes identity data, captures business policy, models roles, and takes a risk-based, proactive approach to managing users and resources. The Open Identity Platform is fully extensible, providing robust analytics which transforms disparate and technical identity data into relevant business information, resource connectivity that allows organizations to directly connect IdentityIQ to applications running in the datacenter or in the cloud, and APIs and a plugin framework to allow customers and partners to extend IdentityIQ to meet a wide array of needs. An open platform allows organizations to build a single preventive and detective control model that supports all identity business processes, across all applications-in the datacenter and the cloud. SailPoint IdentityIQ applies consistent governance across compliance, provisioning and access management processes, maximizing investment and eliminating the need to buy and integrate multiple products.

Password Manager — IdentityIQ Password Manager delivers a simple-to-use solution for managing user passwords across cloud and on-premises applications policies from any desktop browser or mobile device. By providing intuitive self-service and delegated administration options to manage passwords while enforcing enterprise-grade password, IdentityIQ enables businesses to reduce operational costs and boost productivity.

Amazon Web Services (AWS) Governance Module — Enables organizations to extend existing identity lifecycle and compliance management capabilities within IdentityIQ to mission-critical AWS IaaS environments to provide a central point of visibility, administration, and governance across the entire enterprise. This includes policy discovery and access history across all organization accounts, provisioning AWS entities and objects, access review and certification, and federated access support.

SAP Governance Module — Improves the user experience by introducing a new integrated visual interface for navigating and selecting SAP identities and roles as part of IdentityIQ lifecycle management and compliance solution. SAP data is presented in a familiar hierarchy format that closely represents deployed system resources and organizational structures. New filtering capabilities enable more efficient browsing and selection of SAP data so tasks can be performed faster. Improved granular support for separation of duty (SOD) violation policies provides flexibility for customers to craft more detailed identity governance policies that include SAP role details such as T-Codes and Authorization Objects.

Chapter 1: Provisioning with IdentityIQ

The IdentityIQ provisioning capabilities help companies manage system access for their personnel. Provisioning requests can be created and processed in several ways in IdentityIQ, based on the needs and configuration of the installation. In many cases, modifications to access or entitlements you request in IdentityIQ can be automatically reflected in the associated native applications.

This chapter traces the flow of the provisioning plan through its evaluation and preparation for processing into the appropriate native system. Included throughout are the IdentityIQ tasks, business processes and rules that operate on the data as it moves through the process.

Note: Business processes are often referred to as workflows.

At a high level, provisioning requests are processed as follows:

- The provisioning request is made through one of several actions or activities.
- The request is created as a provisioning plan.
- The Provisioning Broker evaluates and compiles the provisioning plan, which often involves dividing the original plan into several partitioned plans. Each partitioned plan addresses a single application.
- Each partitioned provisioning plan is passed to the appropriate handler.
    - For integration configuration or read-write connectors, the change is written to the destination system.
    - For Work Items, a work item is created and assigned to an identity who must for manually process the request into the target system.
- The provisioning actions are confirmed and marked on the identity cube, based on the mechanisms involved.

Use the Administrator Console link, under the gear icon, to access the Provisioning Transactions table to view the status of all provisioning transactions in your implementation of IdentityIQ; connectors, manual work items, and IdentityIQ operations. “Using the Administrator Console” on page 153.

Access to the Provisioning Transaction table is controlled with IdentityIQ rights.

This chapter has the following sections:

Recording Provisioning Requests

You can create provisioning requests in IdentityIQ using any of the following actions or activities:

- “Certifications” on page 4
- “Policy Violations” on page 5
- "Identity-Refresh-Driven Assignments" on page 5
- “Lifecycle Manager Requests” on page 6
- “Lifecycle Event-Driven Provisioning” on page 8
- “Lifecycle Event-Driven Provisioning” on page 8

Provisioning requests create a provisioning plan that the Provision Broker can analyze and process. In all cases, except certification and policy violation-generated requests, provisioning requests create a Workflow case. The Workflow case manages the processing of the provisioning request based on a defined Workflow. See also “Processing Provisioning Requests” on page 10.

Certifications

During a Certification Access Review, certifiers review the system entitlements granted to sets of identities. Access can be approved or revoked for an identity. This certification process can result in:

- **Certificate Remediation** — When an identity’s access to a system is determined to be inappropriate for their job function, the certifier can revoke the entitlement through the Certification Access Review. This process creates a remediation provisioning request in IdentityIQ to remove that access from the source application.
- **Provisioning through Certifications** — When a business role is approved for an identity and that role includes required IT roles the identity does not have, the certifier is prompted to select whether the missing roles must be provisioned for the identity or whether the business role must be approved without provisioning the missing roles. If the certifier elects to provision the missing roles, a provisioning request is created.

  Note: This provisioning option is only presented during the Access Review if the option Enable Provisioning of Missing Role Requirements is selected in the certification specification.

  All revocations and provisioning requests from a specific access review are combined into a single provisioning plan and processed together except in certifications where revocations are processed immediately, such as certifications with the Process Revokes Immediately setting selected.

Policy Violations

Policies defined in IdentityIQ enable the system to evaluate an identity's access or activities and report any inconsistencies with company policies. Violations are reported to the violation owner, often the identity's manager, or the appropriate application owner. The violation owner can then permit an exception or initiate a remediation request. The following types of policy violation remediations are available:

- "Policy Violation Remediations for SOD Policy Violations" on page 5
- "Policy Violation Remediations for Non-SOD Policy Violations" on page 5

Policy Violation Remediations for SOD Policy Violations

Only remediations for role or entitlement Separation of Duties (SOD) violations generate a provisioning request to revoke the invalid access. For example, when a manager evaluates an identity's SOD violations and determines that one of the accesses for the identity must be removed, the manager can request the revocation of the invalid access.

You can create policy violation remediation requests from:

- Policy owner's Policy Violation page that you can from **Manage -> Policy Violations** page.
- Certification on which the violation is noted.

Policy Violation Remediations for Non-SOD Policy Violations

Note: **By default, you cannot remediate non-SOD policy violations with a certification or in the policy violation window.**

You can perform the following actions to enable certification remediate and generate a Work Item:

1. Edit the XML for any policy to include **remediated** as one of its `certificationActions` values to enable certification remediation on that policy type.
2. Select the remediation option for the violation in a certification to automatically create a Work Item that informs the appropriate party of the need to manually correct the violation.

Identity-Refresh-Driven Assignments

You can use the following options on an Identity Refresh task to generate provisioning requests for identities:

- **Refresh assigned, detected roles and promote additional entitlements** — Creates provisioning requests for IdentityIQ to add roles to identity cubes.
- **Provision assignments** — Creates provisioning requests that apply to external applications.

The following table describes these options in more detail:

Option	Description
Refresh assigned, detected roles and promote additional entitlements	Runs the defined assignment rules for roles and examines role detection profiles to update the Assigned and Detected role lists for the identity. **Note: This option does NOT provision access in external system**

Option	Description
Provision assigned roles	Generates provisioning requests to add entitlements required by the currently assigned roles, which can include: - Entitlements for newly assigned roles - Entitlements missing from previously assigned roles. **Note: If a role was previously assigned through an automatic assignment rule and the rule no longer returns true, provisioning requests are generated to remove the entitlements that the role requires. If another assigned role requires those entitlements, they are not removed.**

Note: **By default, the entitlements associated with a role are de-provisioned when the role is removed from an identity. The Disable deprovisioning of deassigned roles option overrides that default and leaves the entitlements intact for the identity while the role is removed.**

Lifecycle Manager Requests

Lifecycle Manager is a separately licensed portion of the IdentityIQ product that is designed to manage entitlements using provisioning requests. Based on their manager status and how the Lifecycle Manager is configured, users can make requests for themselves or for other identities.

In a typical configuration:

- Managers can make requests for their direct reports.
- Help desk users can make requests for themselves and others.
- Any user can make requests for themselves.

Lifecycle Manager Toolbar

When Lifecycle Manager is enabled, the Lifecycle Manager toolbar displays at the top of the IdentityIQ view and supports the following actions:

- “Request Access” on page 7
- "Manage Accounts" on page 7
- "Other Lifecycle Manager Options" on page 8

Note: **The set of identities for which these actions can be taken is based on the individual user’s authority and the Lifecycle Manager configuration. The self-service, Request For Me, options do not include Create Identity.**

Request Access

Request Access includes Role and Entitlement requests. If you are working with a single user, a third tab, **Current Access** displays that you can use to request the removal of Roles or Entitlements. Use the Lifecycle Manager Request Roles feature to generate requests that:

- Add the appropriate role to the specified identities.
- Provision the entitlements the role requires.
- Provision permitted roles, if added to the request when prompted.
- De-provision by removing roles from an identity
  This option generates a provisioning request to remove the role assignment from the identities and the entitlements the role requires if another role does not need the entitlements.

Use the Lifecycle Manager **Request Entitlements** feature to generate requests to:

- Add the entitlement to the specified identity.
- Revoke an identity's current entitlements.
  This option generates a provisioning request that removes the access from the source application or applications.

By default, when you request a new entitlement on an application and the user already has an account on that application, the entitlement is added to the existing account. If needed, you can create a separate account for specific entitlements.

To create multiple accounts for a single identity on an application or to add an entitlement to a specific existing account when several are available:

1. Navigate to the Lifecycle Manager configuration Additional Options page.
2. In the **General Options** section, select an application included in the list for **Applications that support additional account requests**.
3. For the **Account** selection, select the option to create a new account or the option to add the entitlement to an existing account that the identity already has.

Manage Accounts

Use the **Manage Accounts** feature to:

- Request accounts on additional applications — generates provisioning requests.
- Revoke or disable existing accounts — generates provisioning requests.
- Enable disabled accounts — generates provisioning requests to enable or disable accounts.
- Unlock locked accounts — generates provisioning request.

To use the Manage Accounts to request a new account:

1. Navigate to the Lifecycle Manager configuration Additional Options page.
2. In the **Manage Accounts Options section**, select an application included in the list of applications that support account-only requests.
3. For the **Account** selection, select the option to create a new account or the option to add the entitlement to an existing account held by the identity.

   Note: You can also select the Manage Accounts option on the Lifecycle Options page for any group, they can enable, disable, and delete accounts for the existing accounts. The connector must support this action and the action must not be disabled through another setting on the Additional Options page.

Other Lifecycle Manager Options

Other Lifecycle Manager options include the following items:

- **Create Identity** — Creates provisioning plans that update IdentityIQ. You can create a new IdentityIQ identity with a set of attributes that can be configured. The attributes that you can set or change are defined by a form that can be customized. New identities do not have accounts on any application.
- **Edit Identity** — Creates provisioning plans that update IdentityIQ. You can modify attributes for an existing IdentityIQ identity. The attributes that you can set or change are defined by a form that can be customized.

  Note: **Life Cycle Events can cause provisioning outside of IdentityIQ or additional provisioning inside IdentityIQ. In addition. Attribute sync can also cause provisioning outside of IdentityIQ based on create or edit identity.**
- **Manage Passwords** —Resets passwords on target systems which involves a provisioning plan and provisioning action.
- **View Identities** — Does not have provisioning-related functionality and is read-only.

Lifecycle Event-Driven Provisioning

With Lifecycle Manager enabled, Lifecycle Events can be configured in IdentityIQ to represent activities that occur during the normal course of a person’s employment at a company. These activities include events such as joining the company, changing departments or managers, and leaving the company. The shorthand terms for these activities are Joiner, Mover and Leaver.

When Lifecycle Manager is enabled, IdentityIQ contains four pre-defined Lifecycle Events.

Lifecycle Event	Trigger	Business Process Invoked
Joiner	Identity Creation	Lifecycle Event – Joiner
Leaver	Attribute Change: **Inactive** attribute change from `false to true`	Lifecycle Event – Leaver
Manager Transfer	Manager Change	Lifecycle Event – Manager Transfer
Reinstate	Attribute Change: **Inactive** attribute change from `true` to `false`	Lifecycle Event – Reinstate

By default, these events are disabled and must be enabled before the events can be triggered. Lifecycle Events are triggered by specific changes to an identity. These changes can include the following actions:

- Creation
- Manager transfer
- Attribute change
- Complex changes that an `IdentityTrigger` rule detects

The triggered Lifecycle Events invoke business processes, or workflows, that can contain provisioning actions.

Note: **The terms Business Process and Workflow are synonymous. The IdentityIQ user interface refers to these terms as Business Processes which is the term business managers use most often. The IdentityIQ object model and XML use the term Workflows.**

Manage Lifecycle Events and Actions

The Lifecycle Events and the default actions of each of the business process that the pre-defined Lifecycle Events invoke are listed below.

- **Lifecycle Event – Joiner** — Prints the name of the identity to sysout. No actions are taken on the identity. This action is typically modified to provision birthright access for identities.
- **Lifecycle Event – Leaver** — Creates and runs a provisioning plan to disable all accounts the leaving identity has.
- **Lifecycle Event – Manager Transfer** — Prints names of the old and new manager to sysout. No actions are taken on identity or entitlements. This action is typically modified to generate a certification for the new manager to review the access an identity holds. This action can also be used to provision birthright access identified for members of new manager’s group.
- **Lifecycle Event – Reinstate** — Creates and runs a provisioning plan to enable all previously disabled accounts that a returning identity had.

Lifecycle Events and Actions How-To Tasks

You can perform the following tasks for Lifecycle events and actions:

- “How To Edit Pre-defined Lifecycle Events” on page 9
- “How To Create a New a Lifecycle Event” on page 9
- “How To Delete a Lifecycle Event” on page 9

Note: Additional Lifecycle Events and workflows/business processes can be created as needed to support the business needs for each installation.

How To Edit Pre-defined Lifecycle Events

1. Navigate to **Setup -> Lifecycle Events** page.
2. Right-click an entry and click **Edit** or double click an entry.
3. Make desired changes and click **Save**.

How To Create a New a Lifecycle Event

1. Navigate to **Setup > Lifecycle Events** page.
2. Click **Add New Lifecycle Event**.
3. Enter information for **Lifecycle Event Options** and **Behavior**.
4. Click **Save**.

How To Delete a Lifecycle Event

1. Navigate to **Setup > Lifecycle Events** page.
2. Right-click an entry and select **Delete**.

How To Modify Actions for Lifecycle Events

1. Navigate to Navigate to **Setup -> Business Process** page.
2. Select the **Process Designer** tab.
3. Select a process from the **Edit An Existing Process** list

Note: Typically only administrators can edit the identity cube information. This option is available through the Identities > Identities Warehouse.

You can also access IdentityIQ Debug pages and modify actions through the XML Workflow.

See also "Business Process Management" on page 159.

Other Identity Cube Modifications

In addition to the Lifecycle Manager pages, users with the right capabilities can access an administrative interface to make additional identity modifications. Navigate to the **Identities > Identities Warehouse** page.

Most of the information is read-only, but a provisioning plan is generated that updates an identity when you:

- Edit attribute values on the **Attributes** tab.
- Delete or Move account links from the **Application Accounts** tab.
- Change capabilities or assigned scopes on the **User Rights** tab.

If the triggering attributes for the identity have not changed, deleted roles that were assigned by rules are automatically re-assigned to the identity during the next identity refresh. The re-assignment is also processed as an identity-refresh-driven provisioning request.

Processing Provisioning Requests

IdentityIQ creates a master provisioning plan for the requested actions when a provisioning request is submitted from a provisioning request source. A workflow case is also created to manage and track the progress of the provisioning activity. The workflow case contains the workflow that specifies the process to follow.

Note: Certification and policy violation based provisioning does not use workflows.

IdentityIQ ships with pre-defined workflows or business processes which can be customized for each installation as needed. The workflow case created for each provisioning request is associated with the appropriate workflow for the event that generated the request. The following table lists the Workflows that drive the provisioning process from each request source.

Provisioning Request Source	Workflow Invoked
Lifecycle Manager	Main workflows include: LCM Create and Update, LCM Manage Password, LCM Registration and LCM Provisioning
Identity Refresh	Identity Refresh
Define Identities	Identity Update
Lifecycle Events	Each event is managed by the business process listed in **Business Process** field on the Lifecycle Event definition window.

Provisioning Request Source	Workflow Invoked
Certification Remediations / Provisioning	None Managed by and RemediationManager class. **Note: If the certification specifies Process Revokes Immediately, certification starts the remediation process directly.**
Policy Violation Remediations	None Policy violations remediations that certifications create are managed the same as any other certification remediation. Policy violations remediated from **Policy Violations** page are saved directly to the violation table.

Involvement

The **Perform Maintenance** task processes all certification remediation including: roles entitlements and policy violations. This task invokes the Remediation Manager to process the remediation requests.

For certifications with specifications that include the **Process Revokes Immediately** option, the `Certificationer` object invokes the Remediation Manager directly to process the remediation requests. The basic logic of the provisioning process remains the same. The Remediation Manager uses the same mechanisms that the workflows use to complete the requests.

Remediation tasks that are performed on the **Policy Violations** page are not a part of these maintenance task processes.

Note: **Requests on a certification for provision-missing-required-roles are not remediation items. These requests are added to the same provisioning plan as additional actions. The process that manages remediation items also manages these requests.**

Overview of Provisioning Process

The Provisioning Process has three phases:

- "Compiling the Plan" on page 12 — Analysis and preparation of the plan for processing
- "Answering Provisioning Policy Questions" on page 15 — Request of missing required data from a user
- "Implementing the Plan" on page 16 — Submittal of the plan to the appropriate connector to provision the requested access

Compiling the Plan

The Plan Compiler is responsible for the following tasks:

- "Create the Provisioning Project" on page 12 — This task begins with the provisioning plan.
- "Evaluate and Expand Roles" on page 12 — Roles are expanded into entitlement requests.
- "Apply Provisioning Policies" on page 13 — This task a applies the policy which contains list of fields with names that correspond to an application account attribute name the role uses.
- "Identify Questions" on page 15 — This task identifies any missing information for the requests.
- "Filter and Check Dependencies" on page 15 — These tasks streamline the provisioning process and prevent unintended consequences of the requests.
- "Partition the Plan" on page 15 — This task divides the provisioning plan into smaller plans.

Create the Provisioning Project

The provisioning project begins with the provisioning plan. As roles are expanded into entitlement requests, missing information for the requests is identified and the plan is divided into smaller plans. The provisioning project serves as a container for all the smaller plans and includes the following items:

- Original (or master) provisioning plan.
- A set of partitioned plans that contain requests for a single connector.
- An unmanaged plan that contains requests that cannot be processed by any connectors. Items in this plan can be converted to manual work items.

Note: Partitioned plans contains the actions necessary to fulfill the master plan. For example, a single role assignment in the master plan can expand into many entitlement requests in the partitioned plans.

Evaluate and Expand Roles

The master plan is evaluated for any role assignments. If the plan contains role assignments, those roles must be expanded. The role expansion process:

- Identifies IT roles that an assigned business role needs.
- Determines what specific entitlements the IT role needs.
- Adds the entitlements to the lists of account/attribute/permission request for the provisioning project. Each attribute is represented as an Attribute Request or a Permission Request.

For example, Business Role X is added to an identity. Business Role X requires IT Role A which has entitlements associated with its role. The Plan Complier determines that IT Role A is required, identifies the necessary entitlements, and adds the entitlements to the project.

Note: After role expansion is complete, IT Role A does not display in the project. Only the raw entitlements that the IT role A needs are listed.

Apply Provisioning Policies

A provisioning policy is a list of fields with names that correspond to an application account attribute name the role uses. Provisioning Policies can be used to help complete an access request that has unknown data required for provisioning. When a provisioning request requires additional information to complete the access request, you can apply a provisioning policy specified for the application involved. Examples of additional or unknown data that is required for provisioning include the following items:

- Information that is not provided in the original request, such as missing information for a new account or missing information for an additional required role.
- Multiple possible values for a required field.

Types of Provisioning Policies include:

- "Role Provisioning Policies" on page 13 — Removes role uncertainty.
- "Application Provisioning Policies" on page 14 — Applied when a new account is requested.

Role Provisioning Policies

The primary purpose of provisioning policies on roles is to remove any uncertainty for the role. In some cases, examining the role profile can determine the set of entitlements to be provisioned for an IT role. Role profiles can be clear or unclear. When all the role profile terms are joined using AND statements, the profile is clear. IdentityIQ can easily analyze the role profile and provision entitlements that match the profile.

For example, A profile that includes a list of OR terms is unclear, because two or more different `memberOf` values can satisfy the role. The following table provides examples.

Role Profile Example Terms	Type of Terms	Explanation
location='Austin' and memberOf='Engineering'	Profile with a list of AND terms	To satisfy this role, the identity must have both of these account attributes. Requests for those two attributes are added to the plan.
memberOf='Engineering' OR memberOf='Sales'	Profile with a list of OR terms	The default provisioning behavior for profiles containing OR terms is to provision only the first one. In this case, memberOf='Engineering' is added to the plan but not memberOf='Sales'. **Note: If the organization wants memberOf='Sales' provisioned for new role members, a provisioning policy can be defined with one field named memberOf with the field value Sales.**

Note: Fields can also be assigned scripts or rules that enable the appropriate value to be calculated instead of using a hard-coded value.

Application Provisioning Policies

Provisioning Policies can also be specified for applications. These policies are applied when a new account is requested on an application. Application Provisioning Policies are similar to Role Provision Policies and can specify the field values as literal values or through a script or rule. The following actions trigger application provisioning policies:

- Create Account
- Update Account
- Delete Account
- Enable Account
- Disable Account
- Unlock Account
- Change Password
- Create <object type>
- Update <object type>

Application Dependency

You can specify an application dependency at the field level when you create a policy. Application dependency works with synchronous connectors and does not work with connectors that queue plans. Application dependencies are enforced during Create operations. For update and delete, the dependencies are ignored.

Note: IdentityIQ does not undo dependencies during de-provisioning.

To specify an application dependency:

1. Navigate to **Applications -> Application Definition**. On the Provisioning Policies tab of the Application Configuration page select the dependent application for the provisioning.
2. Double-click or right-click the application in the **Application List**.
3. On the Application Configuration page, select the **Provisioning Policies** tab.
4. Define the application dependency at the field level in the **Create Account** and **Create Group** policies.

Application dependency works similar to roles and entitlements. If a dependency is missing, IdentityIQ expands it and executes a Create request for the dependency. If the user has an existing link on a dependent application, IdentityIQ uses the existing link information to derive the value. When there are multiple accounts on the link, the applicable accounts are selected automatically using rules or through an interactive user interface. Selecting an account can be an option to create a new account.

The available attributes are derived from the account schema of all dependent applications. During plan compilation, IdentityIQ reads these properties and determines any new accounts that are required to satisfy the dependency.

During Plan Evaluation, IdentityIQ uses the dependency settings to determine the order that must be used to implement the plan. If a dependency plan fails, all of the dependent plans also fail. If a dependency plan requires a retry, after the retry is successfully completed, the dependent plans are executed. There is special new logic in the Provision with Retries method that loops back to the provisioning step when there are still plans to complete.

There are not transformations (rules) on dependent fields. The evaluation process copies the exact values from the dependency plan or link to the dependent plan.

Identify Questions

After the provisioning policies are applied, pieces of data can still be missing. Some provisioning policies are specifically written so the data must be obtained from a person when the role or application account is requested. These missing data elements are recorded as questions on the provisioning project. These questions are presented to a person who must provide the information necessary to complete the provision request. See “Answering Provisioning Policy Questions” on page 15.

Filter and Check Dependencies

Filter and Check Dependencies streamline the provisioning process and prevent unintended consequences of the requests. During this step of the compilation process:

- The current state of the identity is examined.
- Any entitlements requested in the plan that already exist for the identity are removed from the plan.
- Entitlements that are to be removed, based on a role removal, are examined. This step determines if the identity has another role that requires the entitlement that is scheduled to be removed.

Note: If the identity has another role that requires that entitlement, the entitlement removal request is taken out of the plan.

Partition the Plan

At the end of plan compilation, all the individual entitlement requests identified from the original master plan and the role expansion are partitioned into a set of smaller provisioning plans – one per target. The targets are designated by the connector or **integrationConfig** that IdentityIQ uses to communicate with them. Connections can include:

Note: Any requests in the plan that cannot be processed by any of the integration configurations or read-write connectors are added to the unmanaged plan and are processed manually through IdentityIQ Work Items.

See also "Implementing the Plan" on page 16.

Answering Provisioning Policy Questions

After the plan is compiled, the project can have unanswered questions that must be presented to a person to answer. The provisioning broker does not interface with the user and cannot get answers to these questions. The workflow process, the component that controls the provisioning process, is responsible for getting the questions answered.

Exceptions

Because the following processes can not present forms to users, this interactive provisioning policy phase does not apply for the associated provisioning activities. These requests are only fulfilled if they can be completed with the available information. Because remediation requests are access removal requests, these requests should not require any additional data.

- Processes that manage certification remediations
- Processes that manager provisioning activities
- Policy-violation remediations

Generally projects that have unanswered questions are only an issues if the projects have activities that require a new account to be created for a new assignment or a missing role.

Provisioning Forms

The Lifecycle Manager Provisioning, Identity Refresh, and Identity Update Workflows invoke the Do Provisioning Forms business process. This process presents questions on user-facing forms and collects the answers. The Do Provisioning Forms process separates these actions into the following steps:

- Build Provisioning Form
- Present Provisioning Form
- Assimilate Provisioning Form

Optionally, you can assign owners for individual provisioning policy fields. When an owner is assigned, any questions related to the field are sent to the field owner and not to the access requester. The controlling workflow identifies who receives the questions and then submits the forms to the correct identities.

By default, the Lifecycle Manager Provisioning Workflow contains two opportunities to present provisioning forms to a user, pre-approval and post-approval. The following named steps run the **Do Provisioning Forms** workflow:

- Identity Request Initialize
- Identity Request Provision

A Workflow can have a different number of approval steps between the steps that present provisioning forms. Each approval can modify items in the master plan that cause the project to be recompiled. For example, if an approver rejects one of the role assignments, provisioning questions for an account that role requires might not be needed.

Implementing the Plan

After the plans are partitioned and any missing fields are provided, the subdivided plans can be implemented through one of the following mechanisms:

- “Integrations” on page 17
- “Direct Read-Write Connectors” on page 17
- “Work Items” on page 18

The results are recorded in the plan and indicate if the request was implemented immediately or placed in a queue for future implementation. This status determines when the identity cube is updated to reflect the provisioned changes. See also “Updating the Identity Cube” on page 18.

The following table provides an overview of the provisioning mechanism.

Provisioning Mechanism	Plan Implementation
Integration Executors	Managed plan implementation using integration executors.Starts as an asynchronous process that might not complete immediately.
Direct Read-Write Connectors	Application objects contain the provisioning configuration.
Work Items	Unmanaged plan implementation using the controlling workflow.

Integrations

Integrations are a separately licensed components that communicate with systems within your network. The following table provides and overview of the integration modules and connectors.

System	Module	Connector
Provisioning systems, such as: OIM, ISIM, FIM	Provisioning Integration Modules (PIMs)	Read/write connectors and IntegrationConfigs/Executors
IT Service Management, such as: Remedy, Service Now, HP Service Manager	Service Integration Modules (SIMs)	IntegrationConfigs/Executors
Mobile device management systems, such as: AirWatch, MobileIron, Good Technology	Mobile Integration Modules (MIMs)	Read/write Connectors
IT Security: HP ArcSight	IT Security Integration Module	IntegrationConfigs/Executors
Enterprise Applications: Oracle EBS, SAP Portal, PeopleSoft, Siebel and NetSuite	Enterprise Resource Planning Integration Modules (ERP Integration Modules)	Read/Write Connectors
Mainframe: RACF, CA-Top Secret, CA-ACF2, RACF LDAP and Top Secret LDAP	Mainframe Integration Modules	Read/Write Connectors
Healthcare: Epic and Cerner	Healthcare Integration Modules	Read/Write Connectors
Identity Intelligence/Analytics: SAP GRC	GRC Integration Module	IntegrationConfigs/Executors
Governance platform: Amazon Web Services and SAP	IaaS Governance Modules	Read/Write Connectors

Integration Executors attempt an immediate update of the target application. If the immediate update attempt is unsuccessful, Integration Executors, place the activity in a queue.

Note: **Even if the activity does not immediately commit, the Integration Executors cannot communicate back to IdentityIQ when the request is completed. Therefore, these requests are always considered to be queued.**

See also "Plan Initializer Rule or Script" on page 18.

Direct Read-Write Connectors

Read-write connectors are available to manage data communication between IdentityIQ and an ever-increasing number of applications. For applications using these connectors, you manage provisioning activities through the variables in the Provisioning configuration for that application.

Provisioning using direct read-write connector with these applications is fully automated. These connectors generally:

- Run the plan immediately.
- Can report back a committed status to IdentityIQ in real time.
- Confirm that the changes can be reflected on the identity cube immediately.

See also "Plan Initializer Rule or Script" on page 18.

IdentityIQ Updates

For items that require updates to IdentityIQ, such as roles assigned to an identity or identity attribute changes, a separate plan is created. These requests are similar to direct connector updates. Although no connector is required to complete these internal updates, the requests are run immediately and are reported back as committed when updated.

Work Items

Work Items, opened in IdentityIQ that contain provisioning instructions, to provision unmanaged plans. The controlling workflow or Remediation Manager is responsible for implementing an unmanaged plan. An unmanaged plan:

- Includes provisioning requests to any application where data is aggregated using read-only connectors.
- Does not have an Integration Executor that communicates with the plan.
- Are identified and examined after the Integration Executors and direct read-write connectors are called.

If the unmanaged plan contains any requests, one or more work items are opened in IdentityIQ that contains the provisioning instructions from the plan. Each work item is assigned to a user who is responsible for implementing the changes required to complete the specified provisioning tasks. Work item assignees are often the application or entitlement owner. When the provisioning action is completed, the work item assignee must manually mark the work item as complete.

Note: Provisioning tasks managed through work items are considered queued, rather than committed. Even if the assigned user marks the work item complete, IdentityIQ cannot determine with certainty if the changes were actually made until the next aggregation from the source application is completed.

Plan Initializer Rule or Script

You can specify a Plan Initializer rule or script to run during the implementation of the provisioning plan. An installation-specific rule or script can be added to integration and provisioning configurations. When a rule or script is specified, it runs immediately prior to running the provisioning activity for the application. Provisioning is based on the provisioning plan and application associated configuration or integration executor.

See also “Implementing the Plan” on page 16.

Updating the Identity Cube

Provisioning activities that occur completely within IdentityIQ, such as assigning a business role to an identity, are the only provisioning actions that change the information on the identity cube. For example, implementing a provisioning plan does not update role detections. You must perform an Identity Refresh to update the identity based on the provisioned items. For example, to update the list of detected entitlements and roles, you must perform an Identity Refresh.

Identity Refresh

Provisioning workflows generally includes an Identity Refresh step than can be enabled or disabled as needed for the provision activity. To perform an identity refresh to update the Identity Cube, you must:

- Include an Identity Refresh step in the Workflow, or
- Run an Identity Refresh task after the Workflow completes.

To enable the Refresh step in the workflow the `doRefresh` variable must be set to `True`.

General Guidelines

Direct Read-write connectors — For Direct read-write connectors that process requests immediately, the Identity Refresh step is generally enabled. The changes to application accounts that the connectors make are usually displayed immediately in IdentityIQ.

Queued Requests — Requests that were queued are not applied to the identity cube until a re-aggregation has occurred from the application involved. As a result, the Identity Refresh step is typically disabled for provisioning workflows that are managing integration configuration-driven provisioning activities, because the refresh can not detect any changes until after an aggregation from the source system.

Items that were processed as Work Items from the unmanaged plan are treated as queued requests, because manually closing a Work Item does not necessarily indicate all the work was completed. To confirm that the request was processed, you must perform a re-aggregation from the source system. This aggregation must be followed by an identity refresh to update the identity cube with the information.

Because the **Application Accounts** tab for the Identity Cube displays account data that is recorded on the Link object for the identity, the tab lists the provisioned access immediately following the read-write connector commit or following a re-aggregation from integration configuration-managed applications. However, the entitlement data on the **Entitlement** tab and in any certification is not updated until the Identity Refresh task has run.

Special Case: Optimistic Provisioning

When the workflows are configured for Optimistic Provisioning, provisioned changes appear in IdentityIQ before the changes are confirmed through re-aggregation. Optimistic Provisioning assumes that provisioning requests are completed and then updates the identity cube to display the changes when the request is submitted, not when the request is verified.

Optimistic provisioning configuration is useful for some testing scenarios or product demonstrations, but it is not an ideal configuration for most production environments. Companies often prefer that IdentityIQ indicates a confirmed state of system access and not a desired state.

To configure the workflows for Optimistic Provisioning:

1. Verify that the workflow has the Set the `optimisticProvisioning` process variable. By default, most provisioning-related workflows are configured with this argument
2. Set the optimisticProvisioning process variable, or XML arg, option to True. The default value is false.

   Note: To modify other workflows, add the variable and then follow the steps listed above.

Summary of Workflows, Tasks, and Rules in Provisioning

The following table provides an at-a-glance list of workflows, tasks and rules for provisioning through IdentityIQ.

Type	Name	Purpose / Usage
Workflow	Lifecycle Manager: LCM Provisioning LCM Create and Update LCM Manage Passwords LCM Registration	Manages actions requested through Lifecycle Manager.
Workflow	Identity Update	Manages the provisioning actions required based on an Identity Cube update.
Workflow	Identity Refresh	Manages the provisioning actions required from an Identity Refresh.
Workflow	Lifecycle Event – Joiner Lifecycle Event – Manager Change Lifecycle Event – Leaver Lifecycle Event – Reinstate	Controls the Lifecycle Event-driven activities, which can contain provisioning actions.
Workflow (subprocess)	Do Provisioning Forms	Creates, presents and gathers data from provisioning forms. This step is the interactive provisioning policy phase of provisioning.
Workflow (subprocess)	Do Manual Actions	Presents the unmanaged portion of a provisioning project as work items to be processed manually. Update and Identity Refresh workflows use this step. Lifecycle Manager has a similar step but audits differently.
Workflow (subprocess)	Provision with Retries	Manages retries on the provisioning actions for Lifecycle Manager.
Workflow (subprocess)	Identity Request Initialize Identity Request Violation Review Identity Request Approve Identity Request Approve Identity Changes Identity Request Provision Identity Request Notify Identity Request Finalize Provisioning Approval Subprocess	These workflows subdivide Lifecycle Manager Provisioning into more manageable workflow parts. LIfecycle workflows also use some or all of these tasks.
Task	Identity Refresh	Creates provisioning requests based on application of role assignment rules or role detection.
Task	Perform Maintenance	Processes certification-generated and policy violation-generated remediation requests.

Type	Name	Purpose / Usage
Task	Account Aggregation	Provisioning activities driven by integration configurations or Work Items require a re-aggregation from the target system before the identities can be updated with the access change.
Rule	FieldValue	Identifies the default value for the Provisioning Policy field.
Rule	AllowedValues	Constrains allowed values for the Provisioning Policy field.
Rule	Validation	Defines validation process for Provisioning Policy field.
Rule	Owner	Defines owner for Provisioning Policy field.
Rule	PlanInitializer	Can be specified for any IntegrationConfig or ProvisioningConfig to run installation-specific pre-processing in Plan Evaluation step before carrying out provisioning.
Rule	IdentityTrigger	Can determine the triggering of a Lifecycle Event.

Chapter 2: Forms

Forms are used to present items to users for input in several components of IdentityIQ. They are used with:

- Application and role provisioning policies
- Identity provisioning policy (only applicable for installations using Lifecycle Manager)
- Data entry and approvals in workflow steps
- Present simplified views for process variable and step argument editing in workflows
- Report filter specification

The form implementation and available features varies slightly in these areas, so some features might apply to one use and not to the others, this is noted throughout this document.

For more information about using Forms with IdentityIQ, refer to the forms documents on the SailPoint customer support site or contact your support agent for more information.

Specifying Custom Forms

Form specification is different for each available use. All types of provisioning policies can be specified through the IdentityIQ user interface. In all cases, some of the more advanced and custom forms for workflows can be generated through the Business Process Editor. Some of the more advanced options. however, are only available through subsequent editing of the XML definition. Workflow forms created through the Business Process Editor are embedded within the workflow XML. Workflow forms created through the Business Process Editor are embedded within the workflow XML. Alternatively, they can be defined as independent form objects which can be referenced by multiple workflows, by creating them directly in XML and importing them into IdentityIQ. Report forms must be created as external XML documents and imported into IdentityIQ.

Role/Application Provisioning Policies

Provisioning forms are presented to a user when a provisioning request cannot be completed without user input. The data collection fields that are presented on the form come from the role or application's Provisioning Policy, which is defined by the <Form> element inside the Bundle (role) or Application object's XML. The actual form presented to the user during provisioning of roles or application accounts are system-generated at run-time based on skeleton forms that are pre-defined in IdentityIQ. Requests made through LCM are built with the Identity Update form. Requests that come through the Identity Refresh workflow use the Identity Refresh form. These forms contain a read-only section at the top that displays identifying information about the request, for example, Account ID, First name, Last name. The fields defined in the provisioning policy forms are added to the form at run time, when the form is presented to a user.

Provisioning policy forms define the fields required for the role or application account to be provisioned, often including a default value or script/rule for calculating a value. When a field cannot be calculated by the system during provisioning of an account or role, it must be presented to a user through a form to get the required value. When multiple accounts or roles are part of the same provisioning request, the form might display a collection of fields pulled from various provisioning police forms. On the form, the fields are, by default, grouped in sections according to the application or role to which they belong; this grouping can be overridden, by specifying a section attribute on each of the fields, naming the section into which each field should be placed. See the section attribute description in“Fields” on page 30.

Defining Application Provisioning Policy

An application provisioning policy can be defined for an application on the Provisioning Policies tab of the Application Configuration page, Applications -> Application Definition. Separate policies can be defined for create, update, and delete requests. Additionally, provisioning policies can be specified for creating or updating groups.

The required fields should be specified in the policy with the appropriate field attributes defined. These attributes can include a default value or a script/rule to calculate a default value for the field that can be based on the Identity attributes for the Identity for whom the request is being made. The field **Name** should match the corresponding native attribute on the application. If **Review Required** is selected, the field is always presented on a form during provisioning-request processing, even if a default value is provided or calculated successfully.

For creation-type operations you can specify dependencies between applications and application attributes that imply ordering of the provisioning requests.

Field Properties and Value Properties

The provisioning policy field attributes are grouped into two categories: Field Properties and Value Properties.

Field Properties describe field meta data. This includes the field's name, display name, tool tip help text, type, and owner. It also includes indications of whether the field is single or multi-valued, read-only, hidden, required, or review required. Fields can also be marked with a flag to indicate whether changes to the field value should cause the form to be reloaded. The Read-Only and Hidden attributes can be set to a static value (True or False) or can be defined programmatically through a rule or script. The rule and script options are used to dynamically hide and show the field, or change its edit properties, when the form is reloaded based on changes to values of other fields.

The Value Properties section includes properties specifically related to the field's value. A default value, a set of permitted values, and the field's validation logic can all be set here. The Dynamic attribute determines whether the field's value should be reevaluated on every form reload, when the form is reloaded based on a change in another field's value. It should only be selected when the field's value is rule or script based, such that it might change during the form processing based on other field values entered there.

The default value can be specified as a static value or can be calculated programmatically by a rule or script. In an account creation provisioning policy, an additional option, Dependent, is available as part of the ordered provisioning implementation, which is only available on account create provisioning policies. When the dependent option is selected, an application and attribute must also be selected and the value of the field is set to that attribute value for the Identity. Only applications on which this application is dependent are available for selection here.

The **Allowed Values** list can be specified as a list of values or can be set dynamically by a rule or script. Field validation is optional and can be managed by a reusable rule or with a script.

Defining Role Provisioning Policies

Role provisioning policies are specified through the Role Editor: go to Setup -> Roles, select the role name, and click **Edit Role**. Then click **Add Provisioning Policy** and specify the fields for the policy.

Select the application to which the role provisioning policy applies and then specify the fields for the policy. Fields are specified for role provisioning policies exactly as they are specified for application provisioning policies. Role provisioning policies and application provisioning policies are not the same or to be used interchangeably, however.

Role provisioning is not intended for initial role assignment or for the provisioning of account attributes that are not entitlements. Using role provisioning and application provisioning interchangeably cause conflicts and should be avoided.

Role provisioning is designed to be used for profiles that use complex logic, where it is unclear what should be provisioned or de-provisioned. The role provisioning policy is used to state what to provision, "x and y" or "p and q," and to use the contents of the Identity to make that decision.

Identity Provisioning Policy

The Identity Provisioning Policies are optional forms that can be specified to define the fields that must be provided when an Lifecycle Manager Create or Edit Identity request is submitted. When no Identity Provisioning Policy is defined for the create function, IdentityIQ automatically builds a form that includes the entire set of defined Identity attributes (standard and extended) for the installation. The auto-generated update provisioning policy form contains only identity attributes marked as editable. An Identity Provisioning Policy can be defined to select a subset of those fields, to affect the presentation of those fields, for example, grouping in sections or multi-column layout, or to build in some logic to auto-populate some of the fields.

A third identity provisioning policy also exists to support self-service registrations for IdentityIQ. This form is presented when self-service registration is enabled and a new user requests an IdentityIQ account. The form prompts the user for the information required to create a new user account for the installation.

To create an Identity Provisioning Policy, go to Identity Provisioning Policy of the Lifecycle Manager configuration page. Three policies are available: Create Identity, Update Identity, and Self-service Registration. If a policy has already been defined, the name is displayed. Click the name to open and edit the policy. If no policy has been defined for one of these types, click **Add Policy** to add a new one. Add fields to the policy, defining field attributes as needed on the field definition parallels for an application or role provisioning policy.

Identity Provisioning Policy forms are saved as independent form objects. System Configuration entries (entry key="createIdentityForm", "updateIdentityForm", and "registerForm") point to the appropriate forms for each identity provisioning policy by name. The identity provisioning policy forms are saved as <Form> objects inside the UIConfig attributes map under the keys lcmCreateIdentityProvisioningPolicy and lcmUpdateIdentityProvisioningPolicy on the IdentityIQ Debug pages. These form definitions can be edited directly to implement some of the presentation options, for example, multi-columns or sections. The configurable option available on the user interface do not include these features.

Note: Form features related to the Section attribute (which includes subdividing the form into sections and creating multi-column form configurations) are not supported through the user interface. These must be managed directly in the Form Object XML. Any fields added through the user interface after dividing the form into sections are automatically added to the first section. These fields can be moved to other sections by editing the XML.

Workflow Forms

Several standard work item renderers are provided with IdentityIQ for presenting approvals or other data requests to users. These are written as JSF pages. It is possible to write custom forms in JSF, specifying the JSF page as the renderer for the approval. This is rarely done. Customers who want to use custom forms generally specify these through a Form object.

Forms are used in workflows to present data-gathering pages to a user and define data presentation for approval activities. In many cases, implementations rely on the standard approval work item forms for normal approval actions so do not need to implement custom forms for their approval steps, but they still might choose to use custom forms for non-approval data-gathering activities to which the normal approval forms do not apply. A custom form can be added to a workflow through the Business Process Editor (Setup -> Business Processes) by right-clicking a step and choosing **Add Form** or by adding a form element to a step in the Workflow XML.

Whether the form is specified for an approval or a data-gathering activity, the form element must be embedded within an approval element in the XML. The user interface auto-creates it within an approval element. The workflow XML to specify a custom form looks like this:

```
?xml version='1.0' encoding='UTF-8'?>

<!DOCTYPE SailPoint PUBLIC "sailpoint.dtd" "sailpoint.dtd">

<sailpoint>

<Workflow  explicitTransitions="true"  libraries="Identity"  name="Example Workflow"
type="IdentityLifecycle">

…

  <Step name="Display Form">

    <Approval name="Please enter some data" owner="admin" return="selectedApprover"
send="trace ">

       <Form>

        …        <!-- Form content goes here -->

      </Form>

    </Approval>

  </Step>
```

A custom form can also be created as an independent form object, defined in a separate XML document and imported into IdentityIQ, visible through the console or the Debug pages by viewing the Form objects, and referenced in the approval element as an argument like this:

```
 <Approval…>

     <Arg name='workItemForm' value='Custom Form Name'/>

      …

 </Approval>
```

This option promotes form reuse across workflows. However, these independent form objects cannot be edited through the Business Process Editor like the embedded forms can.

Process Variable and Step Forms in Workflows

While forms added to steps on the Process Designer tab of the Business Process Editor are used to request data required by the process from a user, such as a value for a missing attribute, the process variable and step forms are used to define the information presented on the Basic Views of the Process Variables tab and the Arguments tab of the Step Editor.

These forms are created as an independent form object, defined in a separate XML document and imported into IdentityIQ. They are visible through the console or the Debug pages by viewing the Form objects.

The process variables forms are used to simplify the information displayed on the Process Variables tab by hiding those variables that are rarely, if ever, modified and displaying variables in more logical groups. Changes made in the Basic View are persisted to the Advanced View and more complex configuration can be performed there if needed.

The step forms are referenced from the workflows or stepLibraries. These forms define the form that is presented on the Arguments tab of the Step Editor panel and works similarly to the process variable forms.

Both of these forms are referenced from workflows using the configForm variable. The forms can be defined and viewed and edited on the IdentityIQ debug page.

Report Forms

Report definitions often include a reference to a Form object for displaying the report-specific filter options to the report user. In the Report XML, the form is referenced with a <ReportForm> tag:

```
<ReportForm>

<Reference class="sailpoint.object.Form" id="39535985298ff9839ff98dd" name="My
Custom Report Form" />

</ReportForm>
```

The Form is defined separately in its own XML document and imported into IdentityIQ as a Form object. Each section within the form is created as a separate page in the Edit Report window, where you specify the filters that are applied to the report. The report-specific forms are always merged with the Report Form Skeleton, which defines the Standard Properties and Report Layout pages that apply to every report.

Components of a Form Definition

The basic elements in a Form definition are:

```
<Form>

   <Attributes>(map of name/value pairs that influence the form renderer)

   <Button> (determine form processing actions)

   <Section>(Subdivision of form; may contain nested Sections and Fields)

    <Field>(may contain Attributes map, Script to set value, Allowed Values Definition
script, and Validation Script)
```

Within each of these sections of the form definition, certain attributes might apply to some form uses and not to others. The table below provides a high-level overview of which of the available form elements can be specified for each.

Form Usage	Form Component Availability			
	Field	Button	Section	Field
Application and role provisioning policies (Form)				✓
Identity provisioning policy	✓*	✓*	ü	ü
Workflow approval	ü	ü	ü	ü
Report			ü	ü

* Limitations on the Attribute and Button elements for Identity provisioning policy are discussed in “Attributes” on page 28 and “Buttons” on page 29.

Form

The Form element should contain a single attribute to define the form: a name.

```
<Form name="My Custom Form">
```

If the form is stored in the database as an independent Form object, the name must be unique, no two Form objects can share the same name. This restriction does not apply to Forms defined in-line within a workflow approval step. Name is required for independent Form objects, it is recommended but is not required on in-line forms.

Attributes

Forms can include a map of attributes that are used by the renderer. These are applicable only to workflow forms and, in a limited way, to the identity provisioning policy form.

Attributes are specified with the following keys:

Key	Description
pageTitle	Title to render at top of page, typically larger and a different color than the form title; also displayed in browser window header bar in some cases
title	FormTitle, shown at top of form body
subtitle	Form subtitle, shown below title
readOnly	<entry key="readOnly= "" value="true""/> makes form read-only so the fields are rendered as uneditable text or as disabled HTML components
hideIncompleteFields	hideIncompleteFields="true" hides any fields that do not have all of their dependencies met. Usually only set programmatically to control field presentation in provisioning policy forms, though can be specified in a workflow form XML. This does not create dynamically displayed fields on forms. Fields are not displayed on a form after their dependency values on the same form are entered. Use the new hidden attribute on Fields and Sections to achieve this dynamic display functionality.

Note: The Boolean attributes are only specified if they are true; they default to false when omitted.

Attributes maps are specified as shown here:

```
 <Attributes>
   <Map>
      <entry key="pageTitle" value="Review Non-Employee Request"/>
      <entry key="readOnly" value="true"/>
   </Map>
</Attributes>
```

Attributes do not apply to report forms because the sections in a report form are pulled out of the report's form definition and combined into the Report Form Skeleton for display in the user interface. Even if they are specified in the report form, these attributes are never applied to the resultant form that is displayed to the user.

On an identity provisioning policy form, the pageTitle attribute is ignored because the main page title is programmatically set based on the other action being performed (Create New Identity, Edit Identity Attributes for [Username], or New User Registration). The title and subtitle attributes are displayed in the user interface when specified in the form's attributes map. The readOnly, and hideIncompleteFields function on this form type should not be used because they do not provide useful functionality for this type of form.

Buttons

Buttons enable the user to indicate which action to take next and how to process the data on the form. Buttons only apply to workflow forms. Buttons can be specified on identity provisioning policy forms, but the window does not support any action on them other than next (submit). Since **Submit** and **Cancel** buttons already exist on the window and perform the appropriate functions for the window, additional buttons are unnecessary. They cannot be specified in a role or application provisioning policy form, and they are not used by the report executor when it combines the specific report's form with the Report Form Skeleton.

Buttons require two attributes, a label and an action. The label determines the text displayed on the button. The action determines what the system does in response to clicking that button. There are four available actions:

- **next**: save (and validate fields with validation scripts where specified) any entered form data and set the work item status to approved. This can then be used in the Transition logic to advance the workflow to the next step (OK/Save/Approve/Submit functionality).
- **back**: save entered form data (no validation is performed) and set the work item status to rejected. This can then be used in the Transition logic to return to a previous step or any other appropriate action for a rejection. Saved value is redisplayed on this form if the workflow logic process back through this step again.
- **cancel**: close the form, suspend the workflow and return to previous page in user interface, this leaves work item active. awaiting a different action choice by the user.
- **refresh**: save the entered form data and regenerate the form; not a state transition - just a redisplay of the form (rarely used).

These are examples of button elements.

```
<Button label='Submit' action='next'/>
```

```
<Button label="Cancel" action="cancel"/>
```

Sections

Sections divide a form into logical groupings that are visually marked on the window with boxes around the fields in each section. They can be specified in the XML for all policies except application and role provisioning policies. By default, a separate section is created on the provisioning form for each application (each application's provisioning policy form is rendered in its own section). However, fields in a provisioning policy form can be specified with a section attribute that causes them to be displayed in different sections from the defaults. Sections are treated differently on report forms, each section becomes a separate page on the Edit Report window rather than just a separate section on a contiguous form.

Sections are specified in the form object's XML with a <Section> tag and can be modified by the attributes shown in the table below.

Section Attributes	Description
name	Internal name for section (might be referenced by field objects in some forms).

Section Attributes	Description
label	If non-null, the label is displayed above the section fields in a box on the section border. For report forms, the label is specified in the Edit Report window's sections list. Labels can be specified with text, message catalog keys, or variables (specified with `$(`*`varName`*`)` notation).
type	Rendering type (optional). When no type is specified, fields in the section are editable fields, displayed one field per row, unless the columns attribute specifies otherwise. Other type options are: **datatable**: makes fields in the section non-editable; generally used to display a read-only informational table to give the form user a context for the form's requested data **text**: indicates the section is a block of informational text; if multiple fields are included in a text section, each field is rendered on a separate line with line breaks between them
columns	Number of columns contained in the form section; fields are placed in columns left to right, one field per column before moving to the next row. For example, in a 2 column layout (columns="2"), 4 fields are displayed: Field 1 Field 2 Field 3 Field 4

These are examples of Section elements in the XML for forms.

```
<Section name="authorizations" label="Authorizations" type="datatable">
```

```
<Section columns="2" label="rept_app_section_label" name="customProperties">
```

Sections contain nested field elements and might contain nested sections when sub-groupings are needed.

Fields

Fields are the core element of forms, they are the mechanism by which data gets communicated to and from the user. Fields offer options that affect the appearance or functionality of the field. Some of these are commonly used and others are used very infrequently. Some of these are specified as in-line attributes in the <Field> tag and others are specified as nested elements within the Field.

Field attributes appropriate to all form uses are:

Field Attributes	Description
name	Name for the field that can be referenced in code as the variable name in which the field's value is stored. Avoid using the following field names: accept accept-charset action autocomplete enctype method name novalidate target As well as global attributes: accesskey class contenteditable contextmenu,data-* dir draggable dropzone hidden id itemid itemprop itemref itemscpe itemtype lang spellcheck style tabindex title
displayName	Label for the field; can be text or a message key.

helpKey	Tool tip help text; can be text or a localizable message key.
	
	Example:
	
	`<Field name="firstName" displayName="First Name"`
	`helpKey="Enter the person's first name" />`

Field Attributes	Description
type	Field datatype; influences the display widget used to display the field on a form. **Valid values are:** string, int, long, boolean, date, and SailPoint object types (Identity, Bundle, Permission, Rule, Application), default is string. SailPoint objects are displayed as drop-down lists or combo boxes if multi="true" is also specified. Specifying type="boolean" renders the field as a checkbox; specifying type="date" adds a calendar from which the date can be selected. To pre-select an object in the list, specify the name of the object (not an actual object) as the "value" attribute. Example: `<Field name="role" displayName="Role" type="Bundle" value="TRAKK Basic" />`
multi	Boolean indicating whether the field is multi-selectable. This attribute is only appropriate to drop-down lists, which are then displayed as combo boxes. Used this with SailPoint object field types or with a nested AllowedValues / AllowedValuesDefinition element that populates a selection list for the field. Example: `<Field name="apps" displayName="Applications" type="Application" multi="true"/>`
readOnly	Boolean indicating that the field cannot be edited on the form. The value is displayed as text not in an editable box. Not necessary to specify on fields in a datatable section, since they are already read-only.

Field Attributes	Description
hidden	Boolean that, when true, prevents the field from being displayed on the form. This attribute is used in reporting to make fields available for inclusion on the report detail grid but not actually include them by default. This can be used in any form, but might not be commonly implemented: • In role/application provisioning policies, fields are only shown if the user needs to enter data, so forcing fields to be hidden is not helpful. • In workflows, the hideIncompleteFields attribute on the Form object is more likely to be used with the dependencies attribute on Field to defer field display until dependencies are fulfilled. • In Identity provisioning policy, fields that should be hidden can be omitted from the form instead, however, this could be used for fields that always contain the same value for all users to set that value and suppress the field from the data entry form. For example, `<Field name="status" hidden="true" value="NewHire" /` **Note: Attributes mark hidden are not included in the plan. You must manually add the includeHiddenFields property to the form to include the hidden fields in the plan.** **<entry key="includeHiddenFields" value="true"/>**
required	Boolean indicating whether a value is mandatory for the field; required="true" marks field with * on form to indicate required and prevents form submission without a value for the field Example:`<Field name="myfield" displayName="My Field" required="true"/>`
postBack	Boolean that, when true, causes the form to refresh when the field's data value changes, running any rules or scripts that run on form load `<Field name="application" displayName="Application" type="Application" postBack="true"/>` Supports conditional display/editing of sections or fields based on other field attributes values, automatic population of fields based on other fields, and validation of fields based on actions taken on the form. It only runs when a field loses focus, so it can be used on selection fields or text entry fields.

columnSpan	Used when the section is configured with multiple columns; specifies the number of columns the field should span
	
	`<Section columns="2" label="Identity Info" name="identInfo">`
	
	`<Field name="fname" displayName="First Name" columnSpan="2"/>`

Field Attributes	Description
filterString	Used for fields where type is a SailPointObject class to specify a filter to restrict the set of selectable objects presented in the drop-down list. filterString is specified according to the filter string syntax and should be specified in single quotes so double quotes can be used within the string. Example: `<Field displayName="Role" name="role" type="Bundle" filterString='name.startsWith("TRAKK")' />`

section	Statically defined fields in a form's XML are defined within a section element, so any section attribute specified on those fields is ignored. However, the section attribute can be used to organize fields in an application or role provisioning policy or on a dynamically rendered form.
	
	Application and role provisioning policy forms do not have section elements, so the section attribute can be used to force fields to be grouped differently than the default (default is by application or by role).
	
	Example:
	
	These two fields are put on the form in separate sections, labeled "Important Items" and "Optional Items" respectively.
	
	`<Field name="myField" displayName="My Field" section="Important Items"/>`
	
	`<Field name="optField" displayName="Optional Field" section="Optional Items"/>`
	
	The section attribute on fields is also used in dynamically created forms (such as in Reports where fields are added to the form programmatically through an initialization script). This attribute enables the code to specify the section of the form into which the field should be added.

Field Attributes	Description

displayType	Forces string fields to display as specified, used only for string fields
	
	Valid displayTypes are: radio, combobox, textarea, and label
	
	displayType="radio" and "combobox" are used to override the default display format for permitted-values fields (radio is the default for 2 options while >2 options is rendered as comboBox by default). textarea is used to make a string field display as a text area instead of a regular entry field.
	
	`For label, you can use the field displayName for the text/message key of the label.`
	
	`<Field name="dept" displayName="Department" displayType="radio"`
	
	`<AllowedValues>`
	
	`<String>Accounting</String>`
	
	`<String>Manufacturing</String>`
	
	`<String>Engineering</String>`
	
	`</AllowedValues>`
	
	`</Field>`
	
	`<Field name="comments" displayName="Comments" displayType="textarea" />`

value	Sets the default/initial value for the field. This can be overwritten on the form in most cases as long as the field is not marked readOnly. This is used within sections of type="text" to specify the text to display For application or role provisioning policies, setting a value (whether with this attribute or through a nested <Value>, <RuleRef>, or <Script> element) prevents the field from being included on the form unless reviewRequired is specified since provisioning policies only collect values from a user that they cannot determine or calculate independently. In workflow approvals, value can be specified by string, rule, script, call, or reference (string is default). In reports forms, the value is a reference to the report taskDefinition's input parameter from which to retrieve the starting / default value for the field, for example, value="ref:applications". Example: `<Field name="role" displayName="Role" type="Bundle" value="TRAKK Basic"/>`

Field Attributes	Description
dynamic	This attribute performs two separate functions: (1)For fields with an AllowedValuesDefinition, delays running of allowed values scripts/rules until the field is clicked, instead of running at form load, so it can make use of other data entered on the form instead of just data available on initial form load. (2)During form refresh in response to a value change of a field marked for postBack, only fields marked as dynamic (dynamic="true") have their value scripts/rules re-run; otherwise, the initial value calculated for the field on form load remains in effect as the field's default value

These attributes only apply to the application and role provisioning policies:

Field Attributes	Description
dependencies	List (CSV) of other fields that must be evaluated before this field. Dependencies on the provisioning policy (form) field cause that field to be deferred to a subsequent form that is presented after the form on which its dependencies are presented. The field might also be calculated based on those dependencies instead of presenting it on a later form. This attribute can also be used with dynamic/allowedValues fields. Values of dependencies fields are made available to the allowedValues script or rule, even if the field is presented on a different form.
reviewRequired	Enables a default value to be assigned to the field while still including the field on the form displayed to a user. This enables the default to be edited. If reviewRequired="true" is not specified, provisioning policy form fields with a default value (or value script/rule that returns a value) are omitted from the user-facing form and the default value is automatically used.
authoritative	Boolean that specifies whether the field value should replace the current value rather than be merged with it. Valid for multi-valued attributes only: `<Field name="costCenter" displayName="Cost Center" multi="true" authoritative="true"/>`

Fields can also contain nested elements that help control the display or use of the field.

Nested Elements within Field Elements	Description

Description	Field description, used for XML self-documentation. Not displayed in user interface.
	
	`<Description>`
	`    This field stores the Identity's first name.`
	`</Description>`

Attributes	Attribute map used to control field rendering, specific to the field type. The most common attributes are height and width which are usually specified for textarea fields and for entry boxes that need to be other than the default rendering size. Units for height and width are in pixels
	
	`<Attributes>`
	`   <Map>`
	`       <entry key="height" value="200"/>`
	`       <entry key="width" value="100"/>`
	`   </Map>`
	`</Attributes>`
	
	Two special attributes - xtype and vtype - are discussed in the section below.

Value	Alternative to "value" attribute on <Field>. This is required when specifying complex datatypes such as Map or List.
	
	`<Value>`
	`    <List>`
	`       <String>Thing 1</String>`
	`       <String>Thing 2</String>`
	`    </List>`
	`</Value>`
	
	Also needed for fields of type Date, which are specified as the utime representation of the date:
	
	`<Value>`
	`  <Date>1231971297</Date>`
	`</Value>`
	
	Can be used to specify simpler types like String, Boolean, etc. but not commonly done because value attribute is simpler.

Nested Elements within Field Elements	Description

Script	Script used to initialize the value of the field, alternative to the value element/attribute. Automatically created for fields whose value is set by script through user interface specification.
	
	Example:
	
	`<Script>`
	`   <Source>`
	`      [beanshell code goes here]`
	`return [value or variable that contains value to assign to the field];`
	`   </Source>`
	`</Script>`

RuleRef	Reference to a reusable rule for initializing field value. Alternative to <Script> (and value attribute). Automatically created for fields whose value is set by script through user interface specification.
	
	`<RuleRef>`
	`<Reference class="Rule" name="My Rule" id="402839343985ff930d" />`
	`</RuleRef>`

AllowedValues	Specifies a set of values from which the user can select to assign the field value. Automatically created for fields with an allowed values property set to Value (with a list of values specified) through user interface specification.
	
	`<Field name="dept" …>`
	`   <AllowedValues>`
	`      <String>Accounting</String>`
	`      <String>Manufacturing</String>`
	`      <String>Engineering</String>`
	`   </AllowedValues>`
	`</Field>`
	
	The list renders as radio buttons when only two options exist (and multi is not allowed), as a listbox for more than two options, and as a combobox for multi-selectable fields.

Nested Elements within Field Elements	Description

AllowedValuesDefinition	Populates a list of values from which the user can select a value for the field. This field contains either a <Script> block that specifies the list programmatically or a <RuleRef> that points to a rule containing the beanshell for generating the list. Automatically created for fields with an allowed values property set to Script or Rule through user interface specification.
	
	`<AllowedValuesDefinition>`
	`    <Script>`
	`       <Source>`
	`          import sailpoint.object.*;`
	`          List l = new ArrayList();`
	`          for(WorkItem.State enumItem :`
	`WorkItem.State.values()) {`
	`             List l2 = new ArrayList();`
	`             l2.add(enumItem.toString());`
	`             l2.add(enumItem.getMessageKey());`
	`             l.add(l2);`
	`          }`
	`          return l;`
	`        </Source>`
	`    </Script>`
	`</AllowedValuesDefinition>`
	
	Alternative to AllowedValues element and more commonly used. The list renders as radio buttons when only two options exist (and multi is not allowed), as a listbox for more than two options, and as a combo box for multi-selectable fields.

ValidationScript	Script used to examine and validate the field value entered by the user. The value entered is passed to the validation script in the variable named "value."
	
	`<ValidationScript>`
	`  <Source>`
	`     if (value > 10) {`
	`        return "Value must be less than or equal to 10";`
	`     else`
	`        return null;`
	`   </Source>`
	`</ValidationScript>`
	
	Returns null if no errors and an error message (as string or SailPoint**.tools.message** object) if validation errors exist.

ValidationRule	Reference to reusable rule for field validation. This is the alternative to ValidationScript.
	
	`<ValidationRule>`
	`   <Reference class="Rule" name="My Validation Rule"`
	`id="4028392342f5ff9301" />`
	`</ValidationRule>`

Nested Elements within Field Elements	Description

OwnerDefinition	Used only for application and role provisioning policies to determine the Identity to whom the fields should be presented. This enables specification of a RuleRef,
	script, a Value element or a Value attribute:
	
	`<OwnerDefinition>`
	` <RuleRef>`
	`   <Reference class="rule" name="My Owner Rule" id="4038293483598523" />`
	` </RuleRef>`
	`</OwnerDefinition>`
	`or`
	`<OwnerDefinition>`
	`  <Script>`
	`    <Source>`
	`       import sailpoint.object.*;`
	`Identity myIdentity=context.getObjectByName(Identity,"Walter.Henderson");`
	`       return myIdentity;`
	`    </Source>`
	`  </Script>`
	`</OwnerDefinition>`
	`or`
	`<OwnerDefinition value="IIQApplicationOwner"/>`
	
	Can provide either the string name of owning Identity or the Identity object.
	
	As with Field value, OwnerDefinition value can also be expressed as a nested element. It can be a string Identity name or an Identity object:
	
	`<OwnerDefinition>`
	`  <Value>`
	`   <String>Walter.Henderson</String>`
	`  </Value>`
	`</OwnerDefinition>`
	
	Three special names exist that are translated by IdentityIQ into the appropriate Identity so an OwnerDefinition script is not required for them:
	- IIQParentOwner - owner of the role or application containing the provisioning policy form
	- IIQApplicationOwner - owner of the application associated with the provisioning policy form
	- IIQRoleOwner - owner of the role containing the provisioning policy form
	
	`<OwnerDefinition value=""/>` assigns and presents the field to the access requester.
	
	The user interface offers these options for setting field owners: **Requester** (sets OwnerDefinition to ""), **Application Owner** (sets to "IIQApplicationOwner"), **Role Owner** on Role Provisioning Policies(sets to "IIQRoleOwner"), and **Rule** and **Script** (save as OwnerDefinition with nested RuleRef or Script, as shown above).

Nested Elements within Field Elements	Description

AppDependency	Applies only to application provisioning policies as part of the ordered provisioning function; sets the value for a field based on the value of an attribute on another application on which it is dependent
	`<Field displayName="Login ID" name="login" type="string">`
	`<AppDependency applicationName="LDAP" schemaAttributeName="employeeNumber"/>`
	`</Field>`
	This can only be specified when the application has dependencies declared and can only reference attributes on an application on which the application is dependent. The user interface option for field value named **Dependency** creates this element in the field definition.

Working with the Form Editor

The Form Editor provides a graphical user interface enabling you to create and edit forms without having to edit the xml directly.

The Form Editor contains the following sections:

- Detail View — detailed information about the selected form
- Expandable Tree View — provides an ordered, hierarchical view of the form components
- Edit Options — the available attributes for the selected form item

Detail View

This section displays the detail information about the selected form on clicking the **Details** button. The following table lists displayed attributes for the respective Form Type:

Form Type	Attributes
Application Provisioning Policy Form	Title, Subtitle, Wizard, Owner
Role Provisioning Policy Form	Title, Subtitle, Wizard, Application, Owner
Workflow Form	Title, Subtitle, Wizard

Expandable Tree

The expandable tree section provides an ordered and hierarchical view of the form components.

The tree section can be subdivided into the following components:

- Action buttons — buttons for following actions:
  - **Add Section** — adds section to the expandable tree view
  - **Add Button** — adds button to the bottom of the expandable tree view
    Note: The **Add Button** is applicable to Workflow Approval Forms.
  - **Preview Form** — displays the form layout for all included form components in editor. Helps to preview the form while developing a form to see how it renders on actual operations.
- Components in the tree view — Following are the different components of the tree panel:
  - **Section** — Multiple sections can be added to the tree panel through the **Add Section** button. Using **+** icon Fields and Row with Columns can be added. The Section item can be expanded or collapsed by clicking on them.
    - **Add Field** — Fields can be added under the Section.
    - **Add Row with Columns** — Rows with a maximum of four columns can be added under the Section using the **Choose how many columns in this row** drop down list under the **Edit Options** section.

      Note: When using Rows, the columns attribute of Section and columnSpan attribute of Field would be calculated by Form Editor and existing values would be overwritten.
  - **Button** — All the defined Buttons are added at the end in the tree panel.

Reordering Form Components

Form components can be reordered using drag/drop feature in the following way:

- Sections — sections can be reordered. Sections cannot have sections within them.
- Rows — rows can be reordered within the Section or dragged and dropped into any Section.
- Fields — fields can be reordered within Rows or dragged and dropped into any Section.
- Buttons — can be reordered only within Buttons.

Note: For inappropriate moves of the form components the not allowed icon is displayed.

Edit Options

The Edit Options section on the Editor page displays the attributes that must be modified for the respective actions.

Note: Click on the Apply button after the attributes are modified.

Section

Attributes	Description
Basic	
Name	Internal name for section.

Attributes	Description
Label	Label of Section determines Section text on Edit page. Labels can be specified as text, message catalog keys, or variables (specified with $(variableName) notation).
Subtitle	Section subtitle as description (displayed at the top of the section, above all fields in the section).
Settings	
Hidden	Boolean that, when true, prevents the field from being displayed on the form.
Read Only	Section properties are read only.
Hide Nulls	When set to true, hides fields within the section which have a null value.

Fields and Rows

Attributes	Description
Settings	
Name	Name for the field that can be referenced in code as the variable name in which the fields value is stored
Display Name	Label for the field; can be text or a message key.
Help Text	The text that appears when hovering the mouse over the help icon.
Type	Select the type of field from the drop down list. Select from the following: Boolean — true or false values field. Date — calendar date field. Integer — only numerical values field. Long — similar to integer but is used for large numerical values. Identity — specific identity in IdentityIQ field. Secret — hidden text field. String — text field Application — list of existing application Role — existing type of bundles
Type Settings	
Multi-Valued	Enable this to have more than one selectable value in this field of the generated form.
Refresh On Change	Boolean that, when true, causes the form to refresh when the fields data value changes, running any rules or scripts that run on form reload.

Attributes	Description
Authoritative	Enable to have the field value override the current value rather than merge with it. Applicable only for multivalued attributes.
Required	Boolean indicating whether a value is mandatory for the field; required="true" marks field with * on form to indicate required and prevents form submission without a value for the field.
Read Only	Determine how the read only value is derived: True — value based on the selection from the drop down list Rule — value is based on a specified rule Script — value is determined by the execution of a script
Hidden	Boolean that, when true, prevents the field from being displayed on the form. True — value based on the selection from the drop down list Rule — value is based on a specified rule Script — value is determined by the execution of a script
owner	The owner of the field/row. This is determined by selecting from the following: None — no owner is assigned to this field/row Requester — sets the Owner to this field/row Rule — use a rule to determine the owner of this field/row Script — use a script to determine the owner of this field/row
Value Settings	
Value	Sets the default/initial value for the field/row. This can be overwritten on the form if the field/row is not marked as Read Only. Select None, Value, Rule or Script option.
Allowed Values	Specifies a set of values from which the user can select to assign the field value. Automatically created for fields with an allowed values property set to Value (with a list of values specified) through user interface specification. Select Value, Rule or Script option.
Validation	Ability to specify a script or rule for validating the user's value by selecting None, Rule or Script.

Button

Attributes	Description
Settings	
Action	Action determines what the system does in response to clicking the associated button. Select one from the drop down list: Next — used in the Transition logic to advance the workflow to the next step Back — used in the Transition logic to return to a previous step or any other appropriate action for a rejection. Refresh — save the entered form data and regenerate the form; not a state transition just a redisplay of the form
Read Only	Determines whether to show a button or not on the form renderer.

Attributes	Description
Skip Validation	Determines if client-side required item validation is necessary based on the button clicked by the user. Validation is required if the button is not configured to skip validation, the action is NEXT and there are required items.
Label	Determines the text displayed on the button.
Parameter	Action-parameter of the button.
Value	Action-parameter value.

Form Examples

This section contains examples of XML specifications for the various types of forms.

Application and Role Provisioning Policies and Workflow Forms can all be created through the user interface, though some advanced features might require XML editing to implement. All form types are recorded as XML objects that can be edited through the debug pages as needed. This section reviews the form types in their XML format and shows how they are rendered as a form in the user interface based on that XML definition.

Application and Role Provisioning Policy

Application provisioning policies are specified as <Form> within the <Application> definition. Role provisioning polices are <Form> within the <Bundle> definition. Applications might have more than one provisioning policy form - one for (account) creation, update, and delete provisioning activities plus additional policies for group creation and update. Roles might only have one for role assignment to an Identity.

This sample <Form> definition provides examples of fields slotted into separate sections, assigned to different owners by value or by script, with an permitted values set, and with a validation script. Application provisioning policies are specified within a <Forms> element that wraps all of the specified provisioning policy forms together.

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE Form PUBLIC "iiq.dtd" "iiq.dtd">
<Form name="New Acct Policy" type="Create">
  <Field displayName="Name" name="name" required="true" reviewRequired="true"
type="string">
    <OwnerDefinition value="IIQApplicationOwner"/>
  </Field>
  <Field displayName="Phone" name="phone" required="true" section="Extra Info"
type="string">
    <OwnerDefinition value="IIQApplicationOwner"/>
  </Field>
  <Field displayName="Office Number" name="off_no" required="true" section=""
type="integer">
    <OwnerDefinition>
      <Script>
        <Source>return identity.getManager();</Source>
      </Script>
    </OwnerDefinition>
    <ValidationScript>
      <Source>
   try {
int number=Integer.parseInt(value);
if (number &lt; 100) {
return "Office numbers are all 100 or greater.";
} else{
return null;
}
} catch (NumberFormatException e) {
return "Non-numeric value provided; must be numeric.";
}
</Source>
    </ValidationScript>
  </Field>
  <Field displayName="Region" name="region" required="true" type="string">
    <AllowedValues>
      <String>Americas</String>
      <String>EMEA</String>
      <String>APAC</String>
    </AllowedValues>
  </Field>
```

Application Provisioning Policies can render on multiple forms, depending on the field Owners. Multiple provisioning policy forms can be combined into one form if a request spans multiple applications or roles that each need to gather additional data from the same user.

Identity Provisioning Policy

The XML below creates an identity provisioning policy which implements many of the available form options, including:

The form includes multiple field types (: string, object, and secret -. Secret hides enteredthe text). as it is entered. Object fields are rendered as drop-down list boxes pre-populated with all available items of that type.

- Multi-column configurations
- Multi-column spans for some fields
- Allowed values lists
- Tool tip help prompts
- Field validation (runs when user clicks Submit)
- Filter on object lists for example, show only Manager Identities in Manager drop down list
- Conditional display of sections based on entered field values
- Population of fields based on values entered in other fields

The form includes multiple field types: string, object and secret. Secret hides the text as it is entered. Object fields are rendered as drop-down list boxes pre-populated with all available items of that type.

```
     <?xml version='1.0' encoding='UTF-8'?>

<!DOCTYPE Form PUBLIC "sailpoint.dtd" "sailpoint.dtd">

<Form name="Identity Create Policy" type="CreateIdentity">

  <Description>This is the provisioning policy used when creating a new identity thru
LCM.</Description>

  <Section columns="2">

    <Field displayName="First Name" name="firstname" required="true"
reviewRequired="true" type="string"/>
```

```
    <Field displayName="Last Name" name="lastname" postBack="true" required="true"
type="string"/>
    <Field columnSpan="2" displayName="Username" dynamic="true" helpKey="cube name"
name="name" required="true" type="string">
      <Script>
        <Source>
           if ((null != firstname) &amp;&amp; (null != lastname)) {
               return (firstname + "." + lastname);
           }
           return null;
        </Source>
      </Script>
      <ValidationScript>
        <Source>
            // validation variable comes in as "value"; messages value returned
           // is displayed on screen below field on validation; success should return
            // empty messages list
            import sailpoint.tools.Message;
            import sailpoint.object.Identity;

            List messages = new ArrayList();

            Identity existing =
(Identity)context.getObjectByName(Identity.class,value);
            if (existing == null) {
               // No Identity found with that name, so return empty messages -
               // validation successful
               return messages;
            } else {
               Message msg = new Message();
               msg.setKey("Username: " + value + " already exists. Modify this name
to make it unique.");
               messages.add(msg);
               return messages;
            }
        </Source>
      </ValidationScript>
    </Field>
    <Field displayName="Password" name="password" reviewRequired="true"
type="secret"/>
    <Field displayName="Password Confirmation" name="passwordConfirm"
reviewRequired="true" type="secret"/>
    <Field displayName="Employment Type" displayType="combobox" name="status"
postBack="true" type="string">
      <AllowedValues>
        <String>Employee</String>
        <String>Contractor</String>
      </AllowedValues>
    </Field>
  </Section>
  <Section label="Employee Only Fields">
    <Attributes>
      <Map>
        <entry key="hidden">
          <value>
            <Script>
              <Source>
                 if ("Employee".equals(status)) {
                     return false;
                 } else {
                     return true;
                 }
                </Source>
            </Script>
          </value>
        </entry>
      </Map>
    </Attributes>
    <Field displayName="Manager" filterString="managerStatus == true" name="manager"
type="sailpoint.object.Identity"/>
    <Field displayName="att_email" dynamic="true" name="email" reviewRequired="true"
section="" type="string">
      <Script>
        <Source>
           if (("Employee".equals(status)) &amp;&amp; (null != firstname) &amp;&amp;
(null != lastname)) {
                    return (firstname + "." + lastname + "@demoexample.com");
                }
                return null;
            </Source>
      </Script>
    </Field>
   <Field displayName="Location" name="location" reviewRequired="true" type="string"
value="Austin">
      <AllowedValues>
        <String>Austin</String>
        <String>Brazil</String>
        <String>Munich</String>
        <String>London</String>
        <String>Brussels</String>
        <String>San Jose</String>
        <String>Chicago</String>
        <String>Taipei</String>
        <String>Tokyo</String>
      </AllowedValues>
    </Field>
  </Section>
</Form>
```

Workflow Form

This example XML creates a custom form that displays the Identity's name and asks the user to select a region to which the Identity should be assigned. It demonstrates use of an AllowedValuesDefinition and a ValidationScript as well as Sections of all three types, text, datatable, and default. This form is embedded in the Workflow XML, as it would be if the form were created through the Business Process Editor **Add Form** option. The form could alternatively be created as a standalone form object and referenced as an argument to the approval, as described in “Workflow Forms” on page 25.

```
<Step name="Need Region" posX="359" posY="182">
  <Approval name="Need Region" owner="ref:launcher" return="region"
          send="identityName">
    <Arg name="workItemDescription"
           value="string:Fill in Region for $(identityName)"/>
    <Form>
     <Attributes>
        <Map>
          <entry key="pageTitle" value="Get Region"/>
          <entry key="title" value="Need Region for Identity"/>
        </Map>
     </Attributes>
     <Button action="back" label="Abort"/>
     <Button action="next" label="Submit"/>
     <Button action="cancel" label="Return Item to Inbox"/>
```

```
      <Section name='userInstructions' type='text'>
         <Field value="Employees must be assigned to a region.  Please provide the
correct region for this employee."
/>
      </Section>

      <Section type="datatable">
        <Field displayName="Employee Name" name="identityName"/>
      </Section>

      <Section name="Edit These Fields">
         <Field displayName="Region Value" name="region" required="true"
           type="String">
          <AllowedValuesDefinition>
            <Script>
              <Source>
           import java.util.ArrayList;
    import sailpoint.api.*;
    import sailpoint.object.*;

    List regions = new ArrayList();
    QueryOptions qo = new QueryOptions();

    qo.setDistinct(true);
    qo.addOrdering("region", true);

    List props = new ArrayList();
    props.add("region");

    Iterator result = context.search(Identity.class, qo, props);
    while (result.hasNext()) {
  Object [] record = result.next();
String region= (String) record[0];
regions.add(region);
    }
    return regions;
              </Source>
            </Script>
          </AllowedValuesDefinition>
          <ValidationScript>
              <Source>
    // validation variable comes in as "value"
           import sailpoint.tools.Message;
                List messages = new ArrayList();
                if(value.length() &lt; 6) {
                   Message msg = new Message();
                   msg.setKey("New region must be at least 6 characters.");
                   messages.add(msg);
                }
                return messages;

              </Source>
          </ValidationScript>
      </Field>
    </Section>
  </Form>
```

```
 </Approval>
</Step>
```

Report Forms

Report forms are used to display report-specific filters to the user in the Edit Report window. The form must be created as an independent Form object and referenced from the report definition in a <ReportForm> element.

At run-time, the form is combined with the Report Form Skeleton, which defines the Standard Properties and Report Layout pages. Each section named in the form is created as its own Report Properties page, displayed between the Standard Properties and Report Layout pages. The page name, shown in the **Sections** list and at the top of the form, is specified as the Section's label attribute.

```
   <Form name="Uncorrelated Accounts Report Custom Fields">
      <Section label="Uncorrelated Accounts Parameters" name="customProperties">
<Field displayName="report_input_correlated_apps" filterString="logical==false
&amp;&amp;
authoritative==false"
helpKey="rept_input_uncorrelated_ident_report_correlated_apps"
name="correlatedApps" type="Application" value="ref:correlatedApps"/>
      </Section>
    </Form>
```

An example of a simple report form is shown below. It contains only one section, formatted in two columns with several datatypes represented (dates, objects, and boolean). The displayName and helpKey values on this report are localizable message keys. The values are all pulled from the TaskDefinition's input arguments, if any are provided there, to set the fields' default values.

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE Form PUBLIC "sailpoint.dtd" "sailpoint.dtd">
<Form created="1346776069392" id="4028460239921ba40139921bf510019a"
modified="1346776080142"
name="Application Owner Access Review Report Form">
  <Section columns="2" label="rept_cert_custom_section_title"
name="customProperties">
    <Field displayName="rept_cert_field_create_start"
helpKey="rept_cert_help_create_start"
name="createStartDate" type="date" value="ref:createStartDate"/>
    <Field displayName="rept_cert_field_create_end"
helpKey="rept_cert_help_create_end"
name="createEndDate" type="date" value="ref:createEndDate"/>
    <Field displayName="rept_cert_field_signed_start"
helpKey="rept_cert_help_signed_start"
name="signedStartDate" type="date" value="ref:signedStartDate"/>
    <Field displayName="rept_cert_field_signed_end"
helpKey="rept_cert_help_signed_end"
name="signedEndDate" type="date" value="ref:signedEndDate"/>
   <Field displayName="rept_cert_field_due_start" helpKey="rept_cert_help_due_start"
name="dueStartDate"
type="date" value="ref:dueStartDate"/>
    <Field displayName="rept_cert_field_due_end" helpKey="rept_cert_help_due_end"
name="dueEndDate"
type="date" value="ref:dueEndDate"/>
    <Field displayName="rept_cert_field_apps" helpKey="rept_cert_help_apps"
multi="true"
name="applications" type="Application" value="ref:applications"/>
    <Field displayName="rept_cert_field_tags" helpKey="rept_cert_help_tags"
```

```
multi="true" name="tags"
type="Tag" value="ref:tags"/>
    <Field displayName="rept_cert_field_cert_group"
helpKey="rept_cert_help_cert_group" multi="true"
name="certificationGroups" type="CertificationGroup"
value="ref:certificationGroups"/>
    <Field displayName="rept_cert_field_show_exclusions"
helpKey="rept_cert_help_show_exclusions"
name="exclusions" type="boolean" value="ref:exclusions"/>
  </Section>
</Form>
```

This form is rendered as shown in the **Report Properties** section of the **Edit Report** window.

In report forms, sections can be created without Field definitions, allowing the report's taskDefinition's initialization rule/script to create the form fields. Several of the standard reports, for example, use an initialization rule to generate a pages of Application and/or Identity attribute filters based on the installation's system data, the defined standard and extended attributes, so the report forms themselves are defined with empty sections. The Privileged Access Report form provides an example of a dynamically built form.

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE Form PUBLIC "sailpoint.dtd" "sailpoint.dtd">
<Form created="1346776069939" id="4028460239921ba40139921bf73301b0"
modified="1346776080079"
name="Privileged Access Report Form">
  <Section columns="2" label="rept_priv_access_section_priv_account_attrs"
name="Privileged Account
Attributes">
    <Attributes>
      <Map>
        <entry key="subtitle" value="rept_priv_access_section_instructions"/>
      </Map>
    </Attributes>
  </Section>
  <Section columns="2" label="rept_priv_access_section_account_props" name="Account
Properties">
    <Field columnSpan="1" displayName="rept_identity_roles_field_app"
helpKey="rept_identity_roles_helpN_app" multi="true" name="applications"
type="Application"
value="ref:applications"/>
  </Section>
  <Section columns="2" label="rept_priv_access_section_identity_props"
name="Identity Properties"/>
  <Section columns="2" label="rept_priv_access_section_identity_extended_props"
name="Identity Extended
Properties"/>
</Form>
```

Form Models

Form models are used to simplify that process of passing values between the workflow variables and the form. Form models enable the specification of a Map through which a set of variables can be handed to the form by the workflow. The model is defined in the workflow (or pre-defined model is used), enabling the workflow and form to pass a collection of variables at one time through the specified model. The form renderer is set up to use the model so form fields can name the desired attribute directly without having to reference the model name as well.

Since actions in workflows often center around Identities, a map for the identity object, called IdentityModel, is pre-built in IdentityIQ. A workflow library method - getIdentityModel - can be called by a workflow step to create an IdentityModel map to use in a subsequent step that renders a form. To create an empty map, call this method with no arguments. To pre-populate the map with an identity's current values, specify an identity name or ID as an argument to the step. This method is used with no arguments in the new self-service registration workflow to prepare to create a new identity from the data the user enters on the form.

For an example of the simplification offered by a form model: A workflow form needs to display and permit the user to edit 10 identity attributes. Without form binding, all 10 would have to be defined as individual business process variables and all 10 would have to be sent to and returned from the approval in the workflow. The form would also require all 10 to be defined as form arguments. With a model, the whole Identity can be automatically stored in a single business process variable with a single method call, only one variable (identityModel) must be passed to and returned from the form, and no form arguments need to be defined at all.

Use these steps to use the IdentityModel in a workflow form:

1. Go to Se**tup -> Business Process -> Process Variables** tab.
2. Define a process variable in the workflow (identityModel).
3. In an early step in the workflow, initialize and populate the identityModel by calling the getIdentityModel method in the IdentityLibrary workflow library. Specify the identityModel process variable as the Result Variable for that step.

   To pass an identity name or ID to the method, specify it as an argument to the step (identityName or identityId).
4. In the approval that contains the form, create an argument called workItemFormBasePath and specify the identityModel process variable as its value. The form base path is understood by the form renderer and is automatically applied to permit the form access to the map fields. This enables the passing of the model to and from the form.
5. Reference the components of the identityModel in the form as though they were passed as individual variables, for example, as "firstname", not as "identityModel.firstname".

   Note: When a base path is specified as a form argument, the form renderer assumes all fields on the form are accessed through that base path, so all attributes to be included in the form or returned from it must be included in the model.

   ```
   <Section>
   <Field displayName="user_name" name="name" required="true"
   type="string"/>
   <Field displayName="first_name" name="firstname" required="true"
   type="string"/>
   <Field displayName="last_name" name="lastname" required="true"
   type="string"/>
   <Field displayName="email" name="email" required="true" type="string"/>
   …
   ```
6. (Optional) To provision changes to the identity based on the form, call the buildPlanFromIdentityModel() method in the Identity Library. This examines the versions of the model passed to the form and back from it, identifies differences between them, and creates a provisioning plan to make the required changes.

Refer to the LCM Registration workflow, which ships with IdentityIQ Lifecycle Manager, for a full example of implementing the identityModel.

No other models currently ship with the product, but custom models can be created through some manual coding in the initialization stage.

1. Declare the model variable as a process variable (same as the identityModel), for example appModel.
2. Initialize the model manually, since no library method exists to populate custom models. Instead of a method call in the initialization step, the step executes a script or rule written to populate the desired data into a HashMap that is stored in the custom model variable.
3. Specify the custom model variable as the workItemFormBasePath argument to the workflow's form step.
4. Reference components in the custom model by name in the form. As with identityModel, no reference to the base path should be specified in the form field names.

Identity Model Structure

The IdentityModel map delivered with IdentityIQ contains the following entries:

- all standard Identity attributes and all extended Identity attributes (most as strings; lists when multi-valued)
- detectedRoles (List)
- assignedRoles (List)
- manager (String name, rather than ID
- info map which contains:
    - omanager map (includes ID, name, and displayName of Manager Identity)
- lastRefresh, lastLogin, and passwordExpiration dates
- isWorkgroup, managerStatus, correlated, and correlatedOverriden flag values
- assignedScope name and controlsAssignedScope flag value
- transformerOptions (map of primer identityName or identityId used to populate the IdentityModel)
- class (sailpoint.object.Identity)
- transformerClass (sailpoint.transformer.IdentityTransformer)

Accessing Identity Model Attributes

Any identity model attributes can be displayed on a form or set based on data entered in a form field by supplying the model attribute name as the field name.

Access any single-valued attribute at the top level by specifying its name in the field's name attribute:

```
<Field displayName="first_name" name="firstname" type="string"/>
```

To display the contents of a multi-valued extended identity attribute, use the following syntax. Multi-value extended identity attributes are shown in the identityModel as a list of string values.

```
<Field displayName="Cost Centers" multi="true" name="costcenter" type="string"/>
```

Access any nested attribute, for example, those with a map within the map, using dot notation:

```
<Field displayName="Manager ID" name="info.manager.id" type="string"/>
```

Note: Values in the info map should not be altered through the form, as they will not be updated in the model; they are treated as read-only data that provides supplementary data for the corresponding top-level attribute, and they are automatically refreshed based on updates to that top-level attribute.

Display the contents of an object list in the map, such as assignedRoles, detectedRoles, or workgroups, on a form by creating it as a combo box. Do this by specifying the type as the correct object type and specifying multi="true":

```
<Field displayName="Detected Roles" filterString="type==&apos;it&apos;" multi="true"
name="detectedRoles" type="iiq.object.Bundle"/>
```

```
<Field displayName="WorkGroups" filterString="workgroup == true" multi="true"
name="workgroups" type="Identity"/>
```

The application of a filterString to the workgroups list ensures that only workgroup identities display.

The links list contains a map for each link (account) held by the identity. Access attributes inside that list by referencing the name of the desired link and using dot notation to traverse the map:

```
<Field displayName="App Owner" name="links['HR_Employees'].sys.nativeIdentity"
type="string"/>
```

In development and debugging, it can be helpful to examine the identityModel in XML or as a string representation to clearly see its structure. The identityModel is visible in the workflowCase for any workflow where it is used and is printed to stdout if the trace variable is set to true for the workflow. It can be printed as a string from a workflow step with a System.out.println(identityModel.toString()); statement.

Referencing a Form Model

Form models can be accessed by rules and scripts within workflows or forms within workflows using the `$()` parsing tokens, for example, `$(identityModel.name)`. When variables are referenced with this syntax, the ScriptPreParser expands the short hand path references into the proper `MapUtil.get()` reference. This notation can be used in scripts run from fields, variables, steps, transitions or step actions. The script can explicitly specify the full path to the variable in the model, for example, `$(identityModel.name)`, or it can reference the variable directly when a modelBasePath has been defined, for example `$(name)`.

Note: In order to set a basePath in a form, an argument named modelBasePath (defined as Rule.MODEL_BASE_PATH) must be set declaring the path to be used for all the expanded variables in the form. For example, <Arg name='modelBasePath' value='identityModel'/>. The modelBasePath does not have to be specified at the top level; for example, it can point to a list or map within the top-level map such as <Arg name='modelBasePath' value='identityModel.links[AD]'/> (this is the map representing the user's AD account link).

Note: This $() notation can only be used for retrieving values from the model, it cannot be used to set or change values in the model.

Syntax

Use the following syntax rules when writing references within the scripts:

- A dollar sign with parentheses `[$()]` is used as the parsing token to indicate what contents should be expanded. For example, `$(foo.bar)`.
- Double quotes are valid when enclosing spaces within the variable: `$(foo."bar baz")`. However an expansion token within a quoted string is not processed: `"$(not.expanded)"`.
- Brackets can be used within a variable to access elements in a list: `$(foo.bar[baz=bingo].buzz)` `$(foo.bar[baz="path with spaces"].buzz)`
- When the modelBasePath is set to a sub-map or list within the model, the forward slash escape character (/) can be used to jump to the root of the basePath. This escape character must be the first character after the expansion token. If basePath is set to 'identityModel.links[AD]' and the desired reference is for identityModel.firstname the variable would be written as `$(/firstname)` which would be converted to iiq`.tools.MapUtil.get(identityModel, "firstname")`. Otherwise `$(firstname)` is converted to iiq.tools.MapUtil.get(identityModel, "links[AD].firstname").
- If no basePath is set and the variable only contains a single word, no expansion occurs and a warning is written to the log indicating a possible error condition.

Example Syntax

The following are all valid:

No base path:

- `$(foo.bar)` —> iiq`.tools.MapUtil.get(foo, "bar")`
- `$(foo."bar baz")` —> iiq`.tools.MapUtil.get(foo, "\"bar baz\"")`
- `$(foo.bar[baz="path with spaces"].buzz)`—> iiq`.tools.MapUtil.get(foo, "bar[baz=\"path with spaces\"].buzz")`

Base path = ‘foo’

- `$(foo.bar)` —> iiq`.tools.MapUtil.get(foo, "bar")` (assuming basePath is set to 'foo'. This respects the basePath and does not try to find a "foo" attribute within the "foo" map)
- `$(bar)`—> iiq`.tools.MapUtil.get(foo, "bar")` (assuming basePath is set to 'foo')

Base path = ‘foo.bar[AD]’

- `$(baz)`—> iiq`.tools.MapUtil.get(foo, "bar[AD].baz")` (assuming basePath is set to 'foo.bar[AD]')
- `$(/baz)`—> iiq`.tools.MapUtil.get(foo, "baz")` (assuming basePath is set to 'foo.bar[AD]')

Chapter 3: Configure Risk Scoring

Use the risk scoring configuration pages to define the algorithms used by IdentityIQ to determine risk scores for identities and applications within your organization. Risk scores are used throughout the product to highlight high risk users and accounts and to trigger notices when configured to do so.

To access Risk configuration options, go to **Identities -> Identity Risk Model** or **Application -> Application Risk Model**. Configuring risk scoring requires the assignment of administrative capabilities within IdentityIQ.

To configure risk scoring for identities and applications refer to following:

- "Identity Risk Score Configuration" on page 57
- "Application Risk Score Configuration" on page 61

Identity Risk Score Configuration

IdentityIQ uses a combination of base access risk and compensated scoring method to determine the overall Identity Risk Scores, or Composite Risk Score, used throughout the product. You configure Baseline Access and Composite risk scoring for applications by navigating to the Define > Application Risk Model area of the product interface.

Base access risk is a measure of inherent user access risk. Base risk scores are set on each role, entitlement, and policy defined. This type of score ranges from 0 (lowest risk) to 1000 (highest risk). The account weight assigned to any additional entitlements that are assigned to an identity also have an impact base risk scores. Account weights are factored in to the entitlement baseline access risk scores.

IdentityIQ applies a series of compensating factors to each base risk score to calculate compensated scores. These compensated scores are then weighted using a maximum contribution percentage and combined to form an overall Composite Risk Score for each user.

The compensating factors and weighted values enable IdentityIQ to accurately identify high-risk users based on more than just the roles they are assigned within your enterprise.

For example, a user assigned only low risk roles might be considered high risk if they have never been included in a certification process or the roles they do have are in violation of separation of duty policies.

Scoring Definitions

There are a number of scores, or types of scores, that contribute to the overall Identity Risk Score, or Composite Risk for each IdentityIQ user. The basic scores that are used to determine the overall score are:

Table 1—Access Risk Scoring Definitions

Score	Definition
Base Risk Score	The score assigned to each role, entitlement, or policy violation.
Total Base Risk Score	The total score of all base risk scores of the same component type on a per user basis. For example, add the base risk scores for all roles assigned to a specific user together to determine the role total base risk score.

Table 1—Access Risk Scoring Definitions

Score	Definition
Compensated Risk Score	The value of the base risk score for a component multiplied by the compensating factor for that component type.
Total Compensated Risk Score	The Total Base Risk Score for a specific component type multiplied by the Compensated Risk Score for that component type.
Composite Risk Score or Identity Risk Score	The overall risk score for a user after the composite weighing, or maximum contribution to total score factor, is applied to the total compensated risk scores for each component. The time since the last certification was performed on the user is also figured into this score with the total compensated scores for role, entitlement, and policy violation.

Use the sliding bars or manually enter a value, to define scoring on each panel.

Use the following tabs to create risk score factors for your enterprise:

- Baseline Access Risk Tab — apply base risk scores to roles, entitlements and policy violations. See "Identity Baseline Access Risk Tab" on page 58.
- Composite Scoring Tab — apply compensating factors to base risk scores. See "Identity Composite Scoring Tab" on page 60.

Identity Baseline Access Risk Tab

The Baseline Access Risk score is a measure of inherent risk. A user's Baseline Access Risk score rarely changes because their role within the enterprise is the primary factor in defining the score. This type of score ranges from 0 (lowest risk) to 1000 (highest risk).

Select one of the following options to define how IdentityIQ calculates base access risks. Each role, entitlement, and policy violation is assigned a score that falls into a band. The number of bands is configured on the Advanced Configuration page and applies to the entire IdentityIQ application.

- “Role Baseline Access Risk” on page 58
- “Entitlement Baseline Access Risk” on page 59
- “Policy Violation Baseline Access Risk” on page 59

To configure baseline access risk scores for role, entitlement, and policy violation access, navigate to the Define > Identities Risk Model area of the product interface and select the Baseline Access Risk tab.

Role Baseline Access Risk

Role Baseline Access Risk score is calculated based on the roles correlated to the identity. This list contains every role defined in IdentityIQ. To limit the number of items displayed in the list, filter the list by role name and type.

Table 2— Role Baseline Access Risk Configuration Column Descriptions

Column	Description
Name	The name of the role.
Type	The role type as defined when the role was modeled.
Description	The description of the role as defined when the role was modeled.

Table 2— Role Baseline Access Risk Configuration Column Descriptions

Column	Description
Risk Level	The current risk level assigned to the role.

Click on a role to display the configuration panel to see the role details and set or modify the risk level. Use the slider control to set the risk level or enter a value in the field on the right.

Entitlement Baseline Access Risk

Entitlement Baseline Access Risk score is calculated based on the additional entitlements correlated to an identity. Additional entitlements are entitlements that are assigned to a user, but are not part of any of the roles assigned to that user.

Entitlements fall into two categories: Permissions and Attributes. A Permission is a privilege, such as create, read, update, delete, and execute. Attributes are customized user characteristics made up of an attribute/value pair, such as group/Administrators. A risk score is configured for each Permission and Attribute/Value pair in the system. A user's Entitlement BAR score is determined by summing the risks associated with each of the additional entitlements that they hold.

Use this page to add applications to the list and to work with the entitlements on each. The Entitlement Baseline Access Risk Configuration page contains the following information:

Table 3— Entitlement Baseline Access Risk Configuration Column Descriptions

Column	Description
Application	The name of the application with which the entitlements are associated.
Account Weight	The default score assigned to any identity that is assigned entitlements on this application. Account Weight scores are not compensated. This score is not applied to the identity risk score if the entitlements assigned to the user are, either all used as part of roles assigned to the user, or if the risk score for all of the entitlements assigned to the user are zero based on certification rules.
Permissions	Click in this column to modify the weight assigned to the permissions for the associated application. Use the sliding bar or enter a value in the field on the right to modify permission weight.
Attributes	Click in this column to add, delete or modify the weight assigned to the attributes for the associated application. Select an attribute from the drop-down list, type an attribute name, and click **Add** to assign a weight to a new attribute, or modify and existing attribute in the list. Select an attribute using the check-boxes on the left and click **Delete** to remove an attribute from the list.

To add an application to the list, select an application from the drop-down list on the bottom of the page. The list contains all of the application configure to work with IdentityIQ that are not currently on the list. Use the Permissions and Attributes columns to add entitlements to applications for risk tracking.

Policy Violation Baseline Access Risk

Policy Violation Baseline Access Risk score is calculated using policy violations that are detected for a user based on defined policy rules. A risk score is configured for every rule in each policy or for the policy if no rules apply. This score is calculated by taking the sum of the risks associated with every policy or rule that the user violates.

Use the Policy Violation Baseline Access Risk page to view and modify the risk level associated with each policy or policy rule defined. The page is divided into tables based on policy type. If the policy does not contain rules, set the risk level for the entire policy. Use the slider or type a value in the field to the right.

Identity Composite Scoring Tab

Use the Composite Scoring tab to assign value to the compensating factors for each base component used to calculate the composite risk scores for users. You can also define the maximum contribution of each component to the total score. The maximum composite risk score is 1000. Use the Maximum Contribution to Total Score value to control the impact of compensated scores on composite scores.

Use the Composite Scoring tab to define the maximum impact of a total compensated score on a user's Composite Risk Score. For example, if the time since the last certification on an identity is considered low risk, you can set the Certification Age to a low value, such as 20% so that even at its maximum value that component only contributes 200 points of the total 1000. If, however, policy violations are considered high risk, you can set the Separation of Duty Violation Compensated Score to 100% so that policy violations move users into the high-risk category quickly.Use the Composite Scoring tab to define the maximum impact of a total compensated score on a user's Composite Risk Score.

Table 4— Identity Composite Scoring Configuration

Category	Compensating Control
Role Compensated Score	Based on applying the following compensating factors to each role base score: The user's role has never been certified before The user's role is approved The user's role was allowed as an exception An allowed exception on the user's role has expired Revocation of the user's role is pending Activity monitoring is enabled on one or more applications associated with the user's role
Entitlement Compensated Score	Based on applying the following compensating factors to each entitlement base score: The user's entitlement has never been certified before The user's entitlement is approved The user's entitlement was allowed as an exception An allowed exception on the user's entitlement has expired Revocation of the user's entitlement is pending Activity monitoring is enabled on one or more applications to which the user's entitlement applies

Table 4— Identity Composite Scoring Configuration

Category	Compensating Control
Policy Violation Compensated Score	Based on applying the following compensating factors to policy base score: The user's violation has never been certified before The user's violation was allowed An allowed exception on the user's policy violation has expired The user's policy violation remains uncorrected Activity monitoring is enabled on the applications on which the user's violation occurred
Certification Age Score	Based on applying the following compensating factors to an expired certification: The risk score starts increasing this many days after the latest certification The risk score reaches its maximum value this many days later
Inactive User Score	looks for inactive users. When this score is enabled any identity is found to be inactive, a default risk score of 500 is assigned for this score component

To configure composite risk scoring for identities, navigate to the Define > Identity Risk Model area of the product interface and select the Composite Scoring tab.

Application Risk Score Configuration

IdentityIQ uses a combination of Component and Composite scoring to determine the overall application risk scores used throughout the application. You configure Component and Composite risk scoring for your applications by navigating to the Define > Application Risk Model area of the product interface.

All scores are calculated by first determining the percentage of accounts that have the qualities tested by the component score. For example, if 10 out of 100 accounts are flagged as service accounts, then the raw percentage is ten percent (.10). This number is then multiplied by a sensitivity value which can be used to increase or decrease the impact of the original percentage. The default sensitivity value is 5 making the adjusted percentage fifty percent (.50). This final percentage is then applied to the score range of 1000 resulting in a component score of 500.

After the component score is calculated a weight, or compensating factor, is applied to each component score to determine the amount each contributes to the overall risk score for the application. The resulting score is the composite score. For example, a few violator accounts might increase risk more than many inactive accounts.

To view the currently configured risk information for an application, go to Application Definition page, click on a listed application, and then select the Risk tab.

Use the following tabs to create risk score factors for your enterprise:

- Baseline Access Risk Tab — apply base risk scores to roles, entitlements and policy violations. See "Application Component Scores Tab" on page 62.
- Composite Scoring Tab — apply compensating factors to base risk scores. See "Application Composite Score Tab" on page 62.

Application Component Scores Tab

Use the Component Scores tab to define the values for each account or component.

Service, Inactive, and Privileged component scores look for links that have a configured attribute. For example, the component `service` with a configured value `true`.

The Dormant Account score looks for a configured attribute that is expected to have a date value, for example `lastLogin`. This algorithm has an argument, `daysTillDormant`, that defaults to thirty (30). If the last login date is more than thirty (30) days prior to the current date, the account is considered dormant and is factored into the risk score.

The Risky Account score looks for links whose owning identity has a composite risk score greater than a configured threshold. The default threshold is five hundred (500).

The Violator Account score looks for links whose owning identity has a number of policy violations greater than a configured threshold. The default threshold is ten (10).

Note: If you check Disabled, the component is not used to determine the application risk score.

To configure component risk scoring for applications, navigate to the Define > Application Risk Model area of the product interface and select the Component Scores tab.

Application Composite Score Tab

Use the Composite Scoring tab to apply a weight or compensating factor for each component. Specify the percentage of contribution for the component scores.

To configure composite risk scoring for an application, navigate to the Define > Applications Risk Model area of the product interface and select the Composite Score tab.

Chapter 4: Partitioning

Note: Partitioning is not available on all task or certifications. Partitioning is available for Account Aggregation, Identity Refresh, and Manager Certification generation.

Note: Partitioning is not available on all application types. Partitioning is controlled by both the configuration of the applications you are using and the configuration of the connectors used to communicate with those applications.

Partitioning is used to break operations into multiple pieces, or partitions. Each partition is then placed in a global queue, and machines, or hosts, in a cluster compete to execute the partitions in the queue. Machines are added or removed from the cluster dynamically with automatic balancing. If a machine fails or is taken down while processing a partition, the partition is placed back into the queue and reassigned to a different machine. A single result object is shared by all partitions and is continually updated so you can monitor the overall progress of the partitioned operation. When all partitions have finished executing the result is marked complete.

Each instance of IdentityIQ includes a Server object containing information about what is happening in that instance. For machines running multiple instances of IdentityIQ, you must give each instance must be assigned a unique iiq.hostname and have a unique Server object.

The Server objects include a heartbeat service and is updated by a new system thread on a regular basis. By monitoring server heartbeats, machines in the cluster can detect when another machine fails. When this happens any partitioned requests that were running on that machine are restarted and picked up by a different machine in the cluster, so that failure of one machine does not terminate an entire long running task.

Server objects include some statistics, such as the number of request threads currently active and the request types that are executing. You can view the state of the machines in your cluster on the Administrator Console page. Refer to the *SailPoint IdentityIQ System Administration Guide* for more information.

To activate partitioning you must have applications configured for partitioning, connectors configured to work with those applications, and you must enable partitioning when defining an account aggregation or identity refresh task, or scheduling a manager certification.

- Applications are configured as part of the Account Settings on the Configuration tab of the Application Configuration page, see *SailPoint IdentityIQ Application Configuration Guide*.
- Connector configuration information is located in the latest *SailPoint Integration Guide*.
- Account Aggregation and Identity Refresh information is located in “Account Aggregation” on page 78 and “Identity Refresh” on page 93.
- Scheduling Manager Certification information is located in the *SailPoint IdentityIQ User’s Guide* or the online help.

Configuring Partitioning Request Objects

Partitioning is also maintained using RequestDefinition objects that are defined for each request type. These objects control how each request-type is processed. For example, these objects define the number of threads that run for each request on the instances of IdentityIQ running on a specific machine. The RequestDefinition objects must be defined on each machine, host, in a cluster.

Note: By default the maximum number of threads to run on each host is set to 1. This number can be changed to maximize performance in your environment, but should be done with caution and only after testing and tuning for your environment.

The following RequestDefinition objects are available:

- Aggregation Partition— define the maximum number of threads to run on each host during account aggregations
- Identity Refresh Partition — define the maximum number of threads to run on each host during identity refresh
- Manager Certification Generation Partition — define the maximum number of threads, the error action, and orphan action for partitioned manager certification requests
- Role Propagation Partition — define the maximum number of threads to run on each host during role propagation

To work with the RequestDefinition objects, go to the IdentityIQ Debug page and select RequestDefinition from the **Select an Object** drop-down list.

Chapter 5: Tasks

Note: When working with tasks, do not open multiple tabs or browsers. Opening multiple tabs might cause a change in one tab to overwrite changes made in another.

Task are used to automate the processes which build, update, and maintain the information contained within IdentityIQ. Use the basic tasks provide by SailPoint, or create and customize the task to meet the needs of your organization.

Account aggregation tasks scan applications configured to work with IdentityIQ, discover users and entitlements on those applications, and, optionally, correlate those users and entitlements with roles.

Account group aggregation tasks are used to scan applications and aggregate account groups and application object attributes. These are then used for group certification (either permissions or membership) or for displaying of group information in identity certifications.

Activity aggregation tasks scan applications, discover activity, and then correlate that activity with identities enabling you to track and monitor all activity for possible policy violations.

Activity Alert tasks aggregate and process alerts defined in your system. The aggregation tasks collects active alerts and, either launches an activity processing task to launch the associated actions, or stores the alerts until the next processing task is run.

Identity refresh tasks analyze the information collected for each identity to ensure that it is up-to-date and accurate. Among other things, identity scans can detect and report on policy violations and trigger event certifications.

Application score refresh tasks scan the specified applications and run the configured application scoring algorithms to determine application risk score. These scores are then used to update the information displayed on the Application Risk Scores page.

Missing Managed Entitlement Scan creates any entitlement objects for items added after the application was last aggregated.

Policy Scan tasks evaluate policies against identity cubes and update identity score cards with any policy violations discovered.

Refresh Composite Accounts tasks refresh composite accounts for all identities that could, potentially, have a composite account on the applications selected.

System tasks are configured, by default, to run in the background and perform maintenance, refresh system-wide data, and cleanup old or unused information.

Target aggregation task are used to scan applications for unstructured targets.

Access to components is controlled by IdentityIQ Capabilities and scope. Talk to your system administrator if you need access to additional components.

Tasks Page

The Tasks page contains a list of all of the tasks that have been created. The first time you access the Task page you see the predefined tasks provided by SailPoint. The tasks are grouped into categories based on the task type. You can expand or contract the categories on the grid using the plus (+) or minus (-) icon next to the category name.

Note: Task category headings are only displayed if a task exists in that category.

The task categories are:

- Account Aggregation
- Account Group Aggregation
- Activity Aggregation
- Activity Alerts
- Certification Refresh
- Generic
- Identity
- Scoring
- System
- Target Aggregation

See “Predefined Tasks” on page 66.

Use the search options to limit the number of tasks displayed in the table. Entering a letter, or partial name, in the **Search** field displays any tasks with names containing that letter pattern.

Use this page to create, edit, run, schedule or delete task.

See “Working with Tasks” on page 68.

The Tasks page contains the following information:

Table 5—Tasks Page field descriptions

Field Name	Description
Name	The name of the task as defined when the it was created.
Description	A brief description of the specific task.

Predefined Tasks

SailPoint provides a number of predefined tasks that can be run to aggregate, correlate and refresh information within your enterprise.

Note: The predefined tasks are not templates that can be used to create new tasks. Changes made to these tasks overwrite exiting information. To create new task you must use the New Task drop-down menu at the bottom of the page.

Note: These tasks are defined to perform specific functions within your enterprise. Deleting or altering these tasks might have negative affects on the performance of IdentityIQ.

SailPoint provides the following tasks:

Generic Tasks:

- Refresh Role Indexes — Update all role information and create the indexes needed to perform role searches. You must run this task before performing any role searching.

Identity Tasks:

- Check Active Policies — Scan all users for policy violations and update Identity Risks Scores. Edit this task to specify how policy violations are handled when detected.
- Prune Identity Cubes — Delete identities that have no account links and have no important references. Identities in any of the following states are protected:
    - Marked protected
    - Is a manager (managerStatus flag true)
    - Has capabilities
    - Bundle, Application, Workitem, or TaskResult owners
    - Work item requestor
    - Application secondary owner
    - Application remediator
    - Creator of a MitigationExpiration

  If the **protectIfCertifying** option is on, identities are protected if they are in an active certification. There is also an option to run the scan for analysis but not delete any identities.
- Refresh Entitlement Correlation — Scan all user entitlements and applications to update role assignments.
- Refresh Groups — Scan all users and update the group indexes for all identity groups.
- Refresh Identity Cube — Perform a full refresh of the identity cubes for all users. Edit this task to specify which portions of the identity cubes are refreshed by this task.
- Refresh Risk Scores — Scan all users and update the Identity Risk Scores for each.

Scoring Tasks:

- Refresh Application Scores — Runs the scoring algorithms against all specified applications and updates the Application Risk Scores page.
- Refresh Role Scorecard — Analyzes each role in the system and collects statistics about them.

System Tasks:

- Check Expired Mitigations — Scans all users for temporary exceptions allowed in a certification that have now expired. The original certifier can optionally be notified when allowed exceptions expire.
- Check Expired Work Items — Scans all work items looking for those that need to be canceled or escalated to a different user.
- Complete Orphaned Identity Requests — Removes completed requests for roles that exist in your system.
- Effective Access Index Refresh — Refreshes or rebuilds the effective access index.
- Full Text Index Refresh — Builds and refreshes the index files used for full text searches on defined fields on the access request pages of the Lifecycle Manager. The index files are rebuilt each time this task is run.
- Perform Identity Request Maintenance — Prunes old identity request objects and scans unverified access requests to check for provisioning completeness.
- Perform Maintenance — Prunes identity snapshots, task results, and certifications, escalates orphaned work items, and performs other background maintenance tasks.

Note: Electronically signed objects are not affected by this task.

- Remove Orphan Role Requests — Stops and removes requests for roles that no longer exists in your system. For example, if the sunset date for a role passes before the request is processed, this task removes that request.
- Role Overlap Analysis — Performs impact analysis on a specified role. The task result name is annotated with the name of the selected role so you can tell multiple analysis results apart.
- Synchronize Roles — Synchronizes IdentityIQ roles with the roles on the identity management systems that are configured to work through a provisioning provider.

Working with Tasks

To run or execute a task, right-click on the task name and select **Execute** or **Execute in background**. **Execute** displays a pop-up progress window and opens the Task Result page when it is complete. **Execute in background** launches the task in the background and you must go to the Tasks Results page to track progress or view the finished task.

See "Task Results" on page 73.

Tasks that require sign off generate work items and email notifications that are assigned to the designated signers. Sign off decisions are retained with the task results for tracking purposes.

See "How to Complete Task Work Items" on page 104.

To create a new task, use the **Create new task** drop-down list to select a task type and display the New Task page.

Note: The predefined tasks are not templates that can be used to create new tasks. Changes made to these tasks overwrite exiting information. To create new task you must use the Create New Task drop-down menu at the bottom of the page.

See "How to Create a New Task" on page 68.

To edit an existing task, click a task or right-click and select **Edit** to display the Edit Task page.

See "How to Edit a Task" on page 70.

To schedule a task, right-click and select **Schedule** from the drop-down list to display the New Schedule dialog. You can schedule task to run once, hourly, daily, weekly, monthly, quarterly or annually to meet the requirements of your enterprise and auditors. Go to the Scheduled Tasks tab to view or edit existing schedules.

See "How to Schedule a Task" on page 71 and "Scheduled Tasks" on page 72.

To terminate a currently running task, access the Task Results page, right-click on the task to terminate and select **Terminate** from the drop-down menu. You are asked to confirm the termination request. Task that are currently running are flagged as pending in the Date Complete column of the Task Results table.

To delete a task, right-click the task and select **Delete** from the drop-down menu. Click **Yes** on the confirmation pop-up to delete the task. When you delete a task from the Tasks table, all associated task results are deleted as well.

How to Create a New Task

Use the New Task page to create a task based on the task types provided. Tasks can be as general or specific as required.

See "Task Types" on page 75 for the complete list of tasks types provided.

1. Click or mouse over the Setup tab and select **Tasks** to open the Tasks page.
2. Select a task type from the **New Task** drop-down list to open the New Task page.
3. Enter a **Name** and brief **Description** for the new task. This information is displayed on the Tasks table when the new task is saved.
4. Select a **Previous Result Action** from the drop-down list. **Delete** is select by default.
   Previous result actions determine how subsequent runs of this tasks react to existing task results.

   Delete — overwrite the previous task results with the new information.

   Rename Old — append a numeral to the name of the old task result and preserve both.

   Rename New — append a numeral to the name of the new task result and preserve both.

   Cancel — cancel the new run of the task.
5. ***Optional***: Allow concurrency. Select **Allow Concurrency** to enable two identical tasks to run at the same time.
   If enabled, allow concurreny appends a numeric value to the name of the task that started second.

   If disabled, the second task is canceled and an exception sent to the requestor.
6. ***Optional***: Require sign off.
   a. Select **Required sign off** to expand the Signoff Properties section.
   b. Select an email notification template from the Initial Notification Email drop-down list. For example, the Task Result Signoff template.
   Templates are created and defined when the application is configured.
   c. Specify the escalation criteria for the sign off request. Use the options displayed to set your escalation parameters.
   None — no reminder emails are sent and no escalation is performed for this work item.
   Send Reminders — email reminders are sent at the configured interval.
   Reminders then Escalation — the configured number of reminders are sent and then the work item is escalated to the signers manager.
   Escalation only — this work item is escalated after the configured interval with no reminders being sent.
   d. Specify the required signers.
   Enter the first letter, or letters, of an identity or workgroup to display a selection list of valid identities or workgroups containing that letter string or click the arrow to the right of the field to display all identities and workgroups and select a signer.
   You can add as many signers as required.
7. ***Optional***: Host.
   If you want to choose a specific host or set of hosts to run the task on, add a comma separated list of host names. If multiple hosts are specified, the task manager selects the first active host If there are no active hosts, or if an incorrect host name is given, the task terminates, and an error message is left in the result.
8. ***Optional***: Email task alert.
   Specify the configuration parameters in order to receive the status of different tasks after completion. These settings overwrite the email notification configured at the IdentityIQ Configuration level setting.

- **Email Notification**: Select **Email Notification** to enable the sending of status of task to those recipient whose email is being registered to receive the task status. Use the options displayed to set your notification.

  Disabled — no email notification would be sent.

  Warning — email notification would be sent in case of any warning after completion of task.

  Failure — email notification would be sent in case of task Failure.

Always — email notification would be sent at completion of task irrespective of the task status.

- **Email Notification Template**: (*Applicable only if **Disabled** is not selected*) Select **Task Status** template to send emails on task completion. Templates are customizable.
- **Email Recipients**: (*Applicable only if **Disabled** is not selected*) Select the identity to register them to receive task status notification on emails associated with it.

9. Specify the task options required for the task you are creating. Each task type displays unique task options. **See "Tasks Page" on page 65 for details on each type.**
10. Click **Save** to save the new task to the Tasks table.
    — OR —

    Click **Save and Execute** to save the task to the Tasks table and run it immediately.
    The Tasks Results page displays when the task completes.

    See "Task Results" on page 73.

How to Edit a Task

Use the Edit Task page to make changes to an existing task.

Note: There is no Save As function on the Edit Task page. Any changes made to an existing task overwrite the task you are editing. You must use the Create New Task drop-down menu to create a new task.

Procedure

1. Click or mouse over the Monitor tab and select **Tasks** to open the Tasks page.
2. Click on a task, or right-click on a task and select **Edit** from the drop-down list to open the Edit Task page.
3. Edit the **Name** and **Description** section as needed.

   Note: Changing the name does not save this as a new task and preserve the task being edited. Anything entered here overwrites the existing information.

4. Select a **Previous Result Action** from the drop-down list. **Delete** is select by default.
   Previous result actions determine how subsequent runs of this tasks react to existing task results.

   Delete — overwrite the previous task results with the new information.

   Rename Old — append a numeral to the name of the old task result and preserve both.

   Rename New — append a numeral to the name of the new task result and preserve both.

   Cancel — cancel the new run of the task.
5. ***Optional***: Allow concurrency. Select **Allow Concurrency** to enable two identical tasks to run at the same time.
   If enabled, allow concurreny appends a numeric value to the name of the task that started second.

   If disabled, the second task is canceled and an exception sent to the requestor.
6. ***Optional***: Require sign off.
   a. Select **Required sign off** to expand the Signoff Properties section.
   b. Select an email notification template from the Initial Notification Email drop-down list. For example, the Task Result Signoff template.
      Templates are created and defined when the application is configured.
   c. Specify the escalation criteria for the sign off request. Use the options displayed to set your escalation parameters.
      None — no reminder emails are sent and no escalation is performed for this work item.
      Send Reminders — email reminders are sent at the configured interval.

Reminders then Escalation — the configured number of reminders are sent and then the work item is escalated to the signers manager.
Escalation only — this work item is escalated after the configured interval with no reminders being sent.

d. Specify the required signers.
Enter the first letter, or letters, of an identity or workgroup to display a selection list of valid identities or workgroups containing that letter string or click the arrow to the right of the field to display all identities and workgroups and select a signer.
You can add as many signers as required.

7. ***Optional***: Host.
If you want to choose a specific host or set of hosts to run the task on, add a comma separated list of host names. If multiple hosts are specified, the task manager selects the first active host If there are no active hosts, or if an incorrect host name is given, the task terminates, and an error message is left in the result.
8. ***Optional***: Email task alert.
Specify the configuration parameters in order to receive the status of different tasks after completion. These settings overwrite the email notification configured at the IdentityIQ Configuration level setting.

- Email Notification: Select Email Notification to enable the sending of status of task to those recipient whose email is being registered to receive the task status. Use the options displayed to set your notification.
- Disabled — no email notification would be sent.
- Warning — email notification would be sent in case of any warning after completion of task.
- Failure — email notification would be sent in case of task Failure.
- Always — email notification would be sent at completion of task irrespective of the task status.
- Email Notification Template: (Applicable only if Disabled is not selected) Select Task Status template to send emails on task completion. Templates are customizable.
- Email Recipients: (Applicable only if Disabled is not selected) Select the identity to register them to receive task status notification on emails associated with it.

9. Edit the task options required for the task you are creating.
Each task type displays unique task options.
See “Task Types” on page 75 for details on each type.
10. Click **Save** to save the new task to the Tasks table.
— OR —
Click Save and Execute to save the task to the Tasks table and run it immediately.
The Tasks Results page displays when the task completes.
See “Task Results” on page 73.

How to Schedule a Task

Use the New Schedule dialog to schedule tasks to run during times of low business activity. Schedule recurring tasks as needed to maintain routine compliance within your enterprise.

The New Schedule dialog enables you to assign a unique name and description to the task schedule. This information is stored on the Scheduled Tasks page and displays in the Task Results table.

See “Scheduled Tasks” on page 72 and “Task Results” on page 73.

Procedure

1. Click or mouse over the Monitor tab and select **Tasks** to open the Tasks page.

2. Right-click on a task name and select **Schedule** from the drop-down list to open the New Schedule dialog.
3. Enter a unique name and description for this schedule task.

   Note: **Task that run across time zones run at the time scheduled, relative to the time zone in which they are scheduled. For example, a task scheduled to run at 4:00 PDT runs at 1:00 EDT.**

4. Enter the date and time to launch the first execution of this task.
   You can enter the date manually, or click the ... icon to select a date from the calendar.

   — OR —

   Select the Run Now field to schedule the task to run immediately after clicking Schedule. For recurring task, the task runs at the current time at the specified Execution Frequency.
5. Specify how often this task should run with the **Execution Frequency** drop-down list.
   Subsequent executions of this task occur at the time specified in the First Execution fields.
6. Click **Schedule** to save this scheduled task.
   Go to the Schedule Tasks page to view a list of all scheduled tasks.

   See “Scheduled Tasks” on page 72.

Scheduled Tasks

The Scheduled Tasks page contains a list of all scheduled tasks, whether recurring or one-time only. One-time tasks are removed from the list after they are executed.

Tasks that are scheduled, but do not execute due to malformed task definitions are displayed in the Scheduled Task table with an error icon (!) in the Last Executed column. Tasks that fail in this way never execute and, therefore, never display results on the Tasks Results page. To see details of the execution error, click on the task to display the Edit Schedule dialog. The error information is displayed in the **Last Launch Error** field. Errors of this type should only occur for custom task definitions, not for any of the tasks supplied with the product. To correct the error, delete the task schedule, correct the task definition, and recreate the schedule.

Use the Scheduled Tasks page to edit or delete schedules.

Use the search options to limit the number of tasks displayed in the table. Entering a letter, or partial name, in the **Search** field displays any tasks with names containing that letter pattern. Click **Advance Search** to search by task results.See “Working with Schedules” on page 73.

To create a scheduled task see “How to Schedule a Task” on page 71.

The Scheduled Tasks page contains the following information:

Table 6—Scheduled Tasks Column descriptions

Field Name	Description
Name	The name of the schedule as defined on the New Schedule page.
Next Execution	The date and time at which the task is next scheduled to execute.
Last Execution	The date and time at which the task most recently executed. This field displays an error icon (!) for tasks that do not execute due to malformed tasks definitions.
Last Result	The result of the last run of this task, for example Success or Failed.
Owner	The creator of the schedule.

Working with Schedules

To edit an existing schedule, click a schedule name or right-click and select **Edit** to display the Edit Schedule dialog.

See “How to Edit a Schedule” on page 73.

To delete a schedule, right-click the schedule name and select **Delete** from the drop-down menu. Click **Yes** on the confirmation pop-up to delete the schedule.

How to Edit a Schedule

Use the Edit Schedule dialog

1. Click or mouse over the Monitor tab and select **Tasks** to open the Tasks page.
2. Click on the Schedule Tasks tab to display the list of scheduled tasks.
3. Click a schedule name or right-click and select **Edit** to display the Edit Schedule dialog.
4. Edit the name or description for this scheduled task.

   Note: Task that run across time zones run at the time scheduled, relative to the time zone in which they are scheduled. For example, a task scheduled to run at 4:00 PDT runs at 1:00 EDT.
5. Change the date and time to launch the first execution of this task.
   You can enter the date manually, or click the ... icon to select a date from the calendar.

   — OR —

   Select the Run Now field to execute the task immediately after clicking Schedule. For recurring tasks, the task runs at the current time at the specified Execution Frequency.
6. Specify how often this task should run with the **Execution Frequency** drop-down list.
   Subsequent executions of this task occur at the time specified in the **First Execution** fields.
7. Click **Save** to save this scheduled task and return to the Schedule Tasks page.
   See “Scheduled Tasks” on page 72.

Task Results

The Task Results page contains a list of all of the tasks that have run or are currently running.

Use the search options to limit the number of tasks displayed in the table. Entering a letter, or partial name, in the **Search** field displays any tasks with names containing that letter pattern. Click **Advanced Search** to filter by start date, end date, or results.

Table 7—Task Results Column Descriptions

Column	Description
Name	The name of the task.
Date Complete	The date and time stamp of when the task completed running.

Table 7—Task Results Column Descriptions

Column	Description
Result	The result status, **Pending**, **Success**, or **Failed**. A result of **Success** with an exclamation point (!) indicates that there are warnings associated with the results.
Signoff	The status of the sign off request for the task results. **None** — no sign off required **Waiting** — sign off request not complete **Signed** — a sign off decision has been made
Owner	The name of the user who launched this task.

Click on a task name in the Tasks Results table to display the Task Results page. Each task type returns information specific to the options that were selected. Tasks that executed with partitioning enabled also display the partitioned results, broken down by the host name of the partitions on which they ran.

Several statistics related to task run length are maintained to help identify tasks that are running longer or shorter than expected. Each time a task is run, we save the start time. When the task is complete we calculate the run time in seconds.

These statistics are not set until the task complete. Until then they are zero. The run time change is a positive or negative integer representing the percent change in run length for this task relative to the average at the time was started. A value of 25 means the task ran 25% longer than average, and a value of -10 means the task was 10% faster.

See “Task Types” on page 75 for details on the information that might be on the Task Results page.

To terminate a currently running task, a task flagged as pending in the Date Complete column, right-click on the task and select **Terminate** from the drop-down menu. You are asked to confirm the termination.

To delete task results, right-click on a result and select **Delete**. Tasks that require a sign off can only be deleted by a user with the Signoff Administrator capability.

If a task was scheduled to run but no results were returned, go to the Scheduled Task tab to ensure that errors did not occur during the task execution.

Task Types

The task types are:

- Account Aggregation — scan all applications, discover users and entitlements on those applications, and then correlate those users and entitlements with roles.
  - See “Account Aggregation” on page 78.
- Account Group Aggregation — scans applications and aggregates account groups and application object types. These are then used for group certification (either permissions or membership) or for displaying group information in identity certifications.
  - See “Account Group Aggregation” on page 81.
- Activity Aggregation — scan all applications, discover activity on the applications, and then correlate that activity with identity cubes. This enables you to track and monitor all activity for possible policy violations.
  - See “Activity Aggregation” on page 81.
- Alert Aggregation — scan applications and aggregates alerts from a set of Alert Collectors. These are then used to generate alert actions.
  - See “Alert Aggregation” on page 82
- Alert Processor — process the aggregated alerts against the alert definitions and launch the appropriate action.
  - See “Alert Processor” on page 83
- Application Builder — create multiple IdentityIQ applications or update the attribute map of an existing IdentityIQ application.
  - “Application Builder” on page 85
- ArcSight Data Export — export data for HP ArcSight Database Connector to an external database table.
  - “ArcSight Data Export” on page 87
- Classification — retrieve classification data from File Access Manager and assigns it to entitlements according to correlation logic defined in the applications that aggregate relevant account and group data or in the File Access Manager global configuration settings.
  - “Classification” on page 83
- Data Export — generate a de-normalized data report to export to an external database table.
  - “Classification” on page 83
- Effective Access Indexing — generate an index of any indirect access that was granted through another object. For example a nested group, an unstructured target, or another role.
  - “Effective Access Indexing” on page 84
- Encrypted Data Synchronization Task —re-encrypt data with user-generated encryption key.
  - "Encrypted Data Synchronization Task" on page 90
- Entitlement Role Generator — scans the entitlements in the system and automatically generates a simple role and appropriates a profile for each one that it finds.
  - “Entitlement Role Generator” on page 90
- FIM Application Creator — automatically discover and create FIM Management Agent Applications.
  - “FIM Application Creator” on page 91

- IQService Public Key Exchange — change the public keys that are used for IQService communications
  - "IQService Public Key Exchange" on page 92
- ITIM Application Creator — inspect the IBM Tivoli Identity Manager (ITIM) and retrieve information about the ITIM services (applications). This task auto-generates an application for each service defined in ITIM.
  - "ITIM Application Creator" on page 92
- Identity IQ Cloud Gateway Synchronization — Synchronize the specified objects to the Cloud Gateway.
  - "IdentityIQ Cloud Gateway Synchronization" on page 93
- Identity Refresh — scan all applications, including the IdentityIQ application, to ensure that all identity information is up-to-date and accurate. Refresh identity scans are also used to detect and report on policy violations and trigger event certifications.
  - See "Identity Refresh" on page 93.
- Identity Request Maintenance — scan for completed Lifecycle Manager access requests.
  - See "Identity Request Maintenance" on page 97.
- Missing Managed Entitlements Scan — scan the selected application to create entitlement objects for items added after the application was last aggregated
  - "Missing Managed Entitlements Scan" on page 98
- Novell Application Creator — inspect the Novell IDM application and retrieve information about all connected applications.
  - See "Novell Application Creator" on page 98.
- OIM Application Creator — inspect the OIM application and retrieve information about all connected applications.
  - See "OIM Application Creator" on page 99.
- Policy Scan — runs policies against identity cubes and update identity score cards with any policy violations discovered.
  - See "Policy Scan" on page 99.
- Propagate Role Changes — refreshes identities who have an assigned role whose associated entitlements have changed.
  - "Propagate Role Changes" on page 100.
- Refresh Logical Accounts — is used to refresh composite accounts for all identities that could, potentially, have a composite account on the composite applications selected.
  - See "Refresh Logical Accounts" on page 101.
- Role Index Refresh — updates all role information and creates the indexes needed to perform role searches. You must run this task before performing any role searching.
  - "Role Index Refresh" on page 102
- Run Rule — runs the specified rule with name/value pairs.
  - "Run Rule" on page 102
- Sequential Task Launcher — launches the specified tasks in the order defined. This enables you to launch tasks that must be run sequentially in the proper order without having to schedule each separately based on estimated run times.
  - "Sequential Task Launcher" on page 102

- "System Maintenance" on page 103 — tasks designed to run in the background.
    - See “System Maintenance” on page 103.
- Target Aggregation — scan selected applications for activity targets.
    - See “Target Aggregation” on page 103.

See “Tasks Page” on page 65 for information on working with these task types.

All task types contain the following standard properties:

Table 8—Task Standard Properties

Field	Description
Name	The name of the task as defined when the task was created
Description	Brief description of the task.
Previous Result Action	Previous result actions determine how subsequent runs of this task react to existing task results. **Delete** — overwrite the previous task results with the new information. **Rename Old** — append a numeral to the name of the old task result. **Rename New** — append a numeral to the name of the new task result. **Cancel** — cancel the new run of the task if a task result with the same name exists.
Allow Concurrency	Enable two identical tasks to run at the same time. If enabled, allow concurreny appends a numeric value to the name of the task that started second. If disabled, the second task is cancelled and an exception sent to the requestor.
Require Signoff	Require sign off on the results of this task. Tasks that require sign off generate work items and email notifications that are assigned to the designated signers. Sign off decisions are retained with the task results for tracking purposes.
Host	A comma separated list of host names on which to run this task. If multiple hosts are specified, the task manager selects the first active host If there are no active hosts, or if an incorrect host name is given, the task terminates, and an error message is left in the result.
Number of Runs	The number of times this task has been run.
Average Run Time	The average time in seconds of task runs.
Reset Run Statistics	Reset the statistic if you reconfigure the task and expect the run times to change. When you reconfigure complex tasks like aggregation or refresh, you should consider resetting run statistics. For example, enabling provisioning in the refresh task can profoundly influence run time so statistics should not be diluted by the previous average before provisioning was enabled.
Email Task Alerts	

Table 8—Task Standard Properties

Field	Description
Email Notification	Select a frequency for email notification to be sent upon task completion. **Disable** — no email notification sent on task completion **Warning** — send an email notification if the task results in a warning **Failure** — send an email notification if the task fails **Always** — always send an email notification upon task completion
Email Notification Template	Select a notification email template from the drop-down list.
Email Recipients	The list of users to receive the task completion notification. Use the drop-down arrow to display all identities, or type the first few letters of a name. select names from the list.

Account Aggregation

Account Aggregation tasks scan all applications, discover users and entitlements on those applications, and, optionally correlates those users and entitlements with roles.

Identities that have changed since the last aggregation performed on an application are marked as needing refresh to increase the performance of identity refresh tasks. You can disable this function.

You can perform the correlation functions as part of this task or run account aggregation on all of the applications in your enterprise and then correlate the identity cubes with all of the aggregated information using an identity refresh task.

To perform aggregation on a composite application you must include the composite application and all of the applications that have accounts with which it is associated in the task definition.

Partitioning is available to speed the processing time for account aggregations and level the load on the machines running these tasks. Partitioning is used to break operations into multiple pieces, or partitions. Each partition is then placed in a global queue, and machines, or hosts, in a cluster compete to execute the partitions in the queue. Machines are added or removed from the cluster dynamically with automatic balancing. If a machine fails or is taken down while processing a partition, the partition is placed back into the queue and reassigned to a different machine. A single result object is shared by all partitions and is continually updated so you can monitor the overall progress of the partitioned operation. When all partitions have finished executing the result is marked complete. See, “Partitioning” on page 63.

Note: You must run the Target Aggregation task after this task is complete if you have activity targets set. This tasks removes all targets when it is run.

See “Target Aggregation” on page 103.

The information scanned and updated is determined by the following criteria when the task is created or edited. You can use any combination of options to build a task.

Table 9—Account Aggregator Options

Option	Description
Select an application to scan	The drop-down list of all applications.

Table 9—Account Aggregator Options

Option	Description
Optionally select a rule to assign capabilities or perform other processing on new identities	If accounts are discovered that do not have matching identities in the IdentityIQ application, the rule specified here is used to create a new identity cube. These rules are created during configuration and deployment. Note: Click the "..." icon to launch the Rule Editor to make changes to your rules if needed.
Refresh assigned and detected roles	Scan for newly assigned roles and update identity cubes.
Check active policies	Scan for policy violations and update identity cubes.
Check to updated existing identities, but not to create new identities if a match is not found	Only create links if they can be correlated to an existing identity.
Refresh the identity risk scorecards	Scan for risk score information and update identity risk score cards.
Maintain identity histories	Compare current identity cubes to existing identity cube history, snapshots, and create new snapshots if any changes are discovered.
Enable Delta Aggregation	Enable the connector to aggregate only those accounts that have changed since the last aggregation. This requires support by the connector.
Detect deleted accounts	Compare current aggregated accounts with the accounts previously aggregated and report any deleted accounts. **Maximum deleted accounts**: This is the maximum number of accounts that can be flagged for deletion after an account aggregation. If this number is passed, no accounts are deleted from the application.
Refresh assigned scope	Refresh assigned scope based on changes discovered during the aggregation and correlation process.
Disable auto creation of scopes	Do not automatically assign scope to identities as part of this task.
Disable optimization of unchanged accounts	Use this option to force the aggregation of all accounts, changed or unchanged since the last aggregation.
Promote managed attributes	When enabled, any values for entitlement or permissions encountered while running the task automatically get promoted as managed attributes.

Table 9—Account Aggregator Options

Option	Description
Disable auto-creation of applications	Do not automatically create application objects for multiplexed accounts.
Disable marking the identity as needing refresh	Disable marking only identities on which change was detected as need to be refreshed. All identities are included in subsequent identity refresh task.
Enable Partitioning	Enable partitioning of this task across multiple hosts. **Note:** Partitioning is not supported for PE2 based connectors. Partitioning has to be configured on the applications and connectors before this option is valid.
Terminate when maximum number of errors is exceeded	Terminate after the specified number of errors occurs. If the database is available, the task result contains a message indicating that the task was terminated due to excessive errors. If the database is down, the task result cannot be persisted and the task might appear to remain in the pending state. **Maximum errors before termination** Number of errors to tolerate before terminating the task.
Sequential Execution - Terminate an Error	Force applications to aggregate in the listed order and stop the aggregation task if an error is encountered.
Actions to include in the task result	Select the actions performed as part of the aggregation task for which detailed information should be included in the task results. This task performs a number of individual actions on accounts and identity cubes during the aggregation and configuration processes. By default only the final results of the task are included in the task results report. To included detailed information on the actions performed as part of the task, select those actions from the list. **Correlate Manual** — identities with accounts that were manually correlated. These are not changed by the task. **Correlate Maintain** — correlation information has not changed since the last time this task ran. **Correlate New Account** — a new account was discovered for an existing identity and assigned. **Correlate Reassign** — an existing account was reassigned from one identity to another as part of the correlation process. **Create New Identity** — an account was discovered for an identity that did not exist in IdentityIQ. An identity cube was created for the new identity. **Ignore** — an account for a new identity was discovered, but a new identity cube was not created. This might occur if this tasks is configured to perform correlation only. **Remove Account** — an account discovered as part of a previous aggregation was not found during this aggregation. These accounts are removed from IdentityIQ.

Account Group Aggregation

An Account Group Aggregation task scans applications and aggregates account groups and application object types. These results are then used for group certification (either permissions or membership), for displaying group information in certifications, and for performing identity searches.

The information scanned and updated is determined by the following criteria when the task is created or edited. You can use any combination of options to build a task.

Table 10—Account Group Aggregation Options

Option	Description
Select applications to scan	The drop-down list of all applications configured to work with IdentityIQ.
Filter object types to scan	This option is only available for applications on which multiple application objects can exist. This option is not available if you select to scan more than one application. The list of all object types or account groups associated with the selected application. If nothing is selected, all object types and account groups are included. It might become important to scan object types separately if they share attributes.
Enable Delta Aggregation	Enable the connector to aggregate only those account groups or application objects that have changed since the last aggregation. This requires support by the connector.
Detect deleted account groups	Detect and delete any account group or application object that was deleted on the native application since the last aggregation task was run.
Automatically promote descriptions to this locale	The default locale for the description attribute of the account group or application object. This option is used if an existing description locale is not found.
Description attribute (default description)	**Note: The Description Attribute defined in the Application Group Schema overwrites any value set here.** The attribute that stores the description. This value defaults to the value `description` if this option is not set.
Group Aggregation Refresh Rule	The rule used to set the owner or modify the account group when it is created or refreshed. Click the **...** icon to launch the Rule Editor to modify the rule if needed.
Promote Classifications	Promote classification from the ResourceObject classification to the ManagedAttribute.

Activity Aggregation

Activity Aggregation tasks scan all applications, discover activity on the applications, and then correlate that activity with identity cubes. Using these tasks enables you to track and monitor activity within your enterprise.

The information scanned and updated is determined by the following criteria when the task is created or edited. You can use any combination of options to build a task.

Table 11—Activity Aggregator Options

Option	Description
Select an activity data source	The drop-down list of all activity data sources discovered by IdentityIQ. If no data source is selected, all available data sources are scanned as part of the task. You applications must be configured to support activity tracking.
Enable storage of the last activity position scanned on the data source	If enabled, the task marks the last activity scanned on the data source so that subsequent runs of this task begin scans at that mark instead of rescanning information. If this option is not enabled, each run of the task scans the entire data source.
Store uncorrelated activities so they can be re-scanned and correlated at a later time	If enabled, all activity discovered on the data source is stored, even if that activity does not correlate to a user in the application. Storing this activity enables you to add users and update their identity cubes without having to rescan your data sources.

Alert Aggregation

Alert Aggregation tasks scan applications and aggregates alerts from a set of Alert Collectors. These are then used to generate alert actions.

The information scanned and updated is determined by the following criteria when the task is created or edited. You can use any combination of options to build a task.

Table 12—Alert Aggregation Options

Option	Description
Select sources to scan	The drop-down list of all alert data sources discovered by IdentityIQ. If no data source is selected, all available data sources are scanned as part of the task. You applications must be configured to support alert tracking.
Enable Delta Aggregation	If enabled, the task only aggregates alerts that have occurred since the last run of this task. This option requires support from the connectors being scanned.
Process Alerts	If enabled, an Alert Processor task will launch as soon as this Alert Aggregation task is complete. If you select this option, you can limit the alert types processed in a comma separated list, or process all of the alerts collected.

Alert Processor

Alert Processor tasks process the aggregated alerts against the alert definitions and launch the appropriate action.

The information scanned and updated is determined by the following criteria when the task is created or edited. You can use any combination of options to build a task.

Table 13—Alert Processor Options

Option	Description
Optional filter string to constrain the alerts processed	If not set, all are processed.
Exclude alerts previously processed	Enable to exclude Alerts that were previously processed.
Optional filter string to constrain the alert definitions to match against alerts	If not set, all Alert Definitions are evaluated.
Enable Partitioning	Enable the task to split into partitions and run across multiple threads and hosts, if available.

Classification

The Classification task is used when you are integrating with File Access Manager, to use File Access Manager's classification to flag and categorize entitlements within IdentityQ. This task retrieves classification data from File Access Manager and assigns it to entitlements according to the correlation logic that is defined in the applications that aggregate relevant account and group data, or in IdentityIQ's File Access Manager global configuration settings.

Table 14—Classification Options

Option	Description
Classification Customization Rule	You can use a rule to customize your classification object, for example to add or modify attributes in the object. Rules must be of the type "ClassificationCustomization" to appear in this selection list.
Automatically promote descriptions to this locale	The locale that any description that is included in the File Access Manager objects will be promoted to, by default. This is used if an existing description locale is not found.

After you complete customizing your task options, click **Save** for later use, or **Save and Execute** to save the task and run it immediately.

Data Export

The Data Export task enables you to export IdentityIQ data to an external database. You can select to export any combination of identity, account, and certification data.

Before you can use the Data Export task, you must create the export database tables on your destination data source.

The task schedule user interface includes a button that generates a customized DDL which you can hand off to a database administrator for execution. Once the data source parameters are entered, click Generate Table Creation SQL.

Table 15—Data Export Options

Option	Description
Datasource Parameters	
Database	Select a database type from the drop-down list.
User Name	Enter the user name parameter of the database table.
Password	Enter the password of the database table.
Driver Class	Enter the driver class used for database.
URL	Enter the URL of the database.
Object Export Options	
Export Identities.	Export identity related data. You can perform a full or incremental export. Use the Export Filter field to apply any database filters.
Export Accounts.	Export account related data. You can perform a full or incremental export. Use the Export Filter field to apply any database filters.
Export Certifications.	Export certification related data. You can perform a full or incremental export. Use the Export Filter field to apply any database filters.

After you complete customizing your task options, click Save for later use or Save and Execute to save the task and run it immediately.

Effective Access Indexing

Effective Access is any indirect access that was granted through another object, such as a nested group, an unstructured target, or another role.

Table 16—Effective Access Index Options

Option	Description
Index Entitlement Targets	Include any effective entitlements associated with application that support effective access searching.
Index Role Targets	Include any effective roles associated with application that support effective access searching.
Index direct role permissions	Include any effective direct role permissions associated with application that support effective access searching.

Table 16—Effective Access Index Options

Option	Description
Index direct entitlement permissions	Include any effective direct entitlement permissions associated with application that support effective access searching.
Index unstructured targets	Include any unstructured targets.
Refresh Fulltext Index	Refresh the Fulltext index as part of this task.
Index classifications	Add an entitlement's classifications to the target association that is created when the entitlement target is indexed; in the UI, this means that an entitlement's classifications will be displayed whenever that entitlement occurs as Effective Access.
Promote classifications	Promote classifications up the effective access "chain" to the entitlement that grants the effective access. For example, if EntitlementA grants you effective access to EntitlementB, and EntitlementB has a classification assigned to it, then with the Promote Classifications option enabled, the classification assigned to EntitlementB will also be displayed in the UI for EntitlementA.
Delete all current targets before indexing	Clear an existing Effective Access Index before running this task.

After you complete customizing your task options, click Save for later use or Save and Execute to save the task and run it immediately.

Application Builder

The Application Builder task lets you create multiple IdentityIQ applications, and update existing applications in bulk. The task also includes the ability to perform account and group aggregation for a host using the associated application. It can also export essential data about your existing applications.

The task accepts the inputs required to create or update applications from a `.csv` file. Sample `.csv` files for Linux-Direct and Windows-Local are provided with this task as examples of how input data can be defined. The sample files are located in the `WEB-INF/config` directory of your IdentityIQ installation. You can also use the task's **Read** option to create `.csv` files from your existing applications, to use as models for creating `.csv` files that support the create and update options.

By default, before creating or updating an application on IdentityIQ, a test connection is performed to ensure that the connector is performing correctly. To skip the Test Connection operation, use **Skip Test Connection** in the Application Builder options.

To enable logging for the Application Builder task, add this entry to the `log4j2.properties` file:

```
logger.ApplicationBuilderExecutor.name=sailpoint.task.ApplicationBuilderExecutor
```

```
logger.ApplicationBuilderExecutor.level=debug
```

Before using the task to update an existing application, it is recommended that you use the iiq console to export the application definition, in case you need to restore them to their original state.

When you use this task to **Update** an existing application, the update is partial; that is, the update operation can add new attribute definitions to an existing schema, as well as adding a new schema.

Use the account or group aggregation options to trigger a background aggregation task.

Working with Flexible Schemas and Provisioning Forms

The Application Builder task supports including XML definitions in your csv files if you need to create or update flexible account schemas, or provisioning forms. Refer to the sample `.csv` files provided with this task for examples of how a schema definition can be included in the `.csv` file. Sample files are provided in the `WEB-INF/config` directory for Linux-Direct and Windows-Local.

If your input file includes an XML definition of a Provisioning Form, be aware that importing a Provisioning Form definition in a create or update operation will replace all existing Provisioning Forms with the new form as defined in the `.csv`

Table 17— Attribute Builder Options

Option	Description
Application Type	Select an application type from the drop-down list. This is type of application you want to bulk-process. A single application builder task can only process applications of the same IdentityIQ-supported type, such as JDBC, Active Directory, or LDAP
Operation	Select an operation from the drop-down list. **Create** - create multiple applications by providing parameters in the `.csv` file in the specified format **Update** - update existing applications by providing parameters in the `.csv` file in the specified format **Read** - export existing applications to the `.csv` file format. Any existing exported files will be overwritten if the task is run again using the same filename. **Note: The Read operation reads the attribute map, account schema, and provisioning policy of an existing application present in IdentityIQ and exports it to the file path provided in CSV format. You must provide the application type and file path to which the file is to be exported before running the operation.**

File Path	The file path, including file name, for the `.csv` file. For the **Read** option, this is the path to the location and name of the file the task will create. For **Create** and **Update** options, this is the path to the file containing the data for creating or updating your applications; these files must be present on the application server or accessible within the network.
	
	Sample `.csv` files are provided in the `WEB-INF/config` directory for Linux-Direct and Windows-Local:
	`Application-builder_linux.csv`
	`Application-builder-windows-local.csv`

Account Aggregation	Executes the account aggregation task. The account aggregation task is triggered sequentially. The aggregation task will use the following format; the UID (unique identifier) is generated automatically: `<Application type> + <Account Aggregation> + <Current time stamp> + <UID>`

Table 17— Attribute Builder Options

Option	Description
Group Aggregation	Executes the group aggregation task. The group aggregation task is triggered sequentially. The aggregation task will use the following format; the UID (unique identifier) is generated automatically: `<Application type> + <Group Aggregation> + <Current time stamp> + <UID>`
Number of Applications per Aggregation Task	The number of application included in each aggregation task. Default: 10
Skip Test Connection	Skip the default test connection operation.

ArcSight Data Export

Export data for HP ArcSight Database Connector to an external database table.

The ArcSight data export task enables you to export IdentityIQ data to external tables.

Before you can use the ArcSight data export task, you must create the export databases on your destination data source.

The task schedule user interface includes a button that generates a customized DDL which you can hand off to a database administrator for execution. Once the data source parameters are entered, click Generate Table Creation SQL. The task adds the following tables in database:

Table 18—Tables in database

Tables	Description
sptr_arcsight_export	Table to maintain the task execution history.
sptr_arcsight_identity	Table contains exported data of Identity.
sptr_arcsight_audit_event	Table contains Audit Events information.

Table 19—ArcSight Data Export Options

Option	Description
Datasource Parameters	
Database	Select a database type from the drop-down list.
User Name	Enter the user name parameter of the database table.
Password	Enter the password of the database table.
Driver Class	Enter the driver class used for database.
URL	Enter the URL of the database.
Object Export Options	

Table 19—ArcSight Data Export Options

Option	Description
Export Identities	Export Identity related data in ArcSight tables. It provides the following options: Full: Exports all the records irrespective if they were exported earlier. Incremental: Exports only records that are updated since last run of this task. This option can even be selected when running the task for first time. When the task is running for first time, this option exports all records similar to the Full option.
Export Audits	Export Audit Events in ArcSight table. It provides the following options: Full: Exports all the records irrespective if they were exported earlier. Incremental: Exports only records that are updated since last run of this task. This option can even be selected when running the task for first time. When the task is running for first time, this option exports all records similar to the Full option.

After you complete customizing your task options, click Save for later use or Save and Execute to save the task and run it immediately.

Configuring HP ArcSight Task to populate host name or IP

The value of column application_host can be populated by adding a map with the value as arcsightAppNameHostMap as shown in the following example. The fieldThis is read from the map as explained below:

It is difficult to determine the host name or IP address of the account as the field is not constant in Application definition in IdentityIQ. Hence, customer can define a map in TaskDefinition and select the task added to export data in ArcSight table. The key in the map should be name of the application defined in IdentityIQ and value should be hostname, IP, or any string that ArcSight administrator understands.

To add the map:

1. Go to debug page, navigate to TaskDefinition and open the ArcSight task configured above.
2. Add the entry as key = Name of Application defined in IdentityIQ and value as the string to identify host of Account like Hostname or IP.
3. Save the task definition. For example:

```
  <entry key="arcsightAppNameHostMap">
    <value>
        <Map>
          <entry key="LinuxApp1" value="linux01.iiq.com"/>
          <entry key="LinuxApp2" value="127.15.19.21"/>
          <entry key="ADDirectApp" value="AD.iiq.com"/>
          <entry key="ServiceNowApp" value="https://iiq.service-now.com"/>
          <entry key="ACF2App" value="ACF2-Mainframe"/>
        </Map>
    </value>
</entry>
```

Note: If the application name is not defined in the map the host field is blank.

Following fields are added in export table:

Table 20—IdentityIQ sptr_arcsight_identity export table

Fields	Description
linkid	Primary key for Link table in IdentityIQ database. This field is copied from iiq_link table id field and is the primary key for export table.
identityid	Primary key in Identity table. This field is copied from iiq_Identity table.
modified_dt	Populates timestamp when the record is exported in export table. The field can be referred while configuring time based ArcSight database connector.
identity_display_name	Represents Display Name of Identity which is copied from iiq_identity table field (display_name).
identity_firstname	Represents first name of Identity which is copied from iiq_identity table field (firstname).
identity_lastname	Represents last name of Identity which is copied from iiq_identity table field (lastname).
application_type	Populates the type of Account which is connected to the Identity like ActiveDirectory – Direct, ACF2 – Full, Box, Cloud Gateway, ServiceNow and so on.
application_host	The host name, IP, or any string which can be used by ArcSight administrator to identify the host of link/account uniquely. Customer can enter any string which can be sent to ArcSight to identify the host of link. This field can be populated as explained in "Configuring HP ArcSight Task to populate host name or IP" on page 88.
application_name	Populates the name of Application of the Account connected to the Identity.
link_display_name	The account connected to the identity which is copied from iiq_link table, field display_name.
entitlements	Represents comma separated list of entitlements to the link of Identity.
risk_score	Represents the composite risk score of Identity.

Table 21—IdentityIQ sptr_arcsight_audit_event export table

Fields	Description
auditid	The audit ID which is primary key for the export Audit table. The field is copied from iiq_audit_event table id field.
created_dt	Populates timestamp when the record is exported in export table. The field can be referred while configuring time based ArcSight database connector.
owner	Describes the Owner of the audit generated.
source	Provides more details to help ArcSight administrator determine the source of audit.
action	Describes the action taken on entity.

Table 21—IdentityIQ sptr_arcsight_audit_event export table

Fields	Description
target	Provides target details.
application	Describes the name of application the target belongs to.
account_name	The name of Account is populated in this field.
attribute_name	The name of attribute modified.
attribute_value	The value provided to the attribute.

Encrypted Data Synchronization Task

The Encrypted Data Synchronization Task is used to re-encrypt IdentityIQ data when a new custom encryption key is generated.

Table 22—Encrypted Data Synchronization Task Options

Option	Description
Disable Application Synchronization	Select this option to ignore encryption key synchronization against applications.
Disable Identity Synchronization	Select this option to ignore encryption key synchronization against identities.
Disable IntegrationConfig Synchronization	Select this option to ignore encryption key synchronization against IntegrationConfig objects.
Disable Attachment Synchronization	Select this option to ignore encryption key synchronization against attachments.
Convert Encrypted Identity Secrets to Hashing	Select this option to convert any encryption keys to use hashing.

Once you have completed customizing your task options, click Save for later use or Save and Execute to save the task and run it immediately

Entitlement Role Generator

The Entitlement Role Generator creates an Entitlement Role for every entitlement found in a specified application. Recommended role types are Entitlement or IT.

You can further refine creation by specifying an entitlement name or permission target so that only entitlements matching the specified criteria are used.

It is recommended to specify a template to be used to name the created roles. IdentityIQ uses Velocity templates. If no template is used, a generic name based on either the entitlement or role is created.

Table 23—Entitlement Role Generator Options

Option	Description
Applications	Select one or more applications from the drop-down list.
Type of Role to Create	Input the name of the role based on the specifications for your enterprise.
Enter the locale to check for descriptions. (If left blank the default Locale is used)	Enter the location of the role description.
Generate entitlements from attributes whose name starts with	Enter letters in the attribute name to filter the scan.
Generate entitlements from permissions whose target starts with	Enter letters in the permission name to filter the scan
Velocity template from which to generate entitlement role names. The template is always passed the applicationName parameter. The description, attributeName, attributeValue, permissionTarget, and/or permissionRights parameters are set when available.	Enter the Velocity template string.

FIM Application Creator

Run the FIM Application Creator task automatically discover and create FIM Management Agent Applications. This task auto-generates an application for each management agent defined in FIM. The applications generated by this task are added to the list of applications in IdentityIQ.

Table 24—ITIM Application Creator Options

Option	Description
Microsoft Forefront Identity Manager Applications	Select applications to inspect and from which applications should be generated based on the management agents found.
Native Object Types of Account	Specify the native object types of accounts created by this task.
Native Object Types of Group	Specify the native object types of groups created by this task.

IQService Public Key Exchange

Run the IQService Public Key task to change the public keys that are used for IQService communications.

Table 25—ITIM Application Creator Options

Option	Description
Select IQService based application(s)	Select IQService based applications on which to change the public keys.

ITIM Application Creator

Run the ITIM Application Creator task to inspect IBM Tivoli Identity Manager (ITIM) and retrieve information about the ITIM services (applications). This task auto-generates an application for each service defined in ITIM. Each ITIM application contains a list of services that are roughly equivalent to the list of applications maintained in IdentityIQ. The applications generated by this task are added to the list of applications in IdentityIQ.

Table 26—ITIM Application Creator Options

Option	Description
ITIM Applications	Select applications to inspect and from which applications should be generated based on the services found.
Generated application name prefix	Specify a prefix to append to any applications created by this task.
Generated application name suffix	Specify a suffix to append to any applications created by this task.

IdentityIQ Cloud Gateway Synchronization

IdentityIQ Cloud Gateway Synchronization tasks scan selected IdentityIQ applications for specified objects and synchronizes them with IdentityIQ Cloud applications. It is intended for use when IdentityIQ is not able to communicate directly with the managed system.

Table 27—IdentityIQ Cloud Gateway Synchronization

Option	Description
IdentityIQ Cloud Gateway application name	Select the name of the application to synchronize.
Applications hosted on the IdentityIQ Cloud Gateway	Select the name of the hosted cloud gateway application with which to synchronize the IdentityIQ application.
Rules to be executed on the IdentityIQ Cloud Gateway	Select which rules to execute against selected applications.

Identity Refresh

Refresh identity tasks scan all identities to ensure that all identity information is up-to-date and accurate. Refresh identity scans are also used to detect and report on policy violations and launch event certifications.

Incremental identity refresh can be configured to only refresh those identities on which information has changed since the last refresh was performed, to increase performance.

Note: Partitioning is disabled if you enable Mark dormant scopes after refresh or Refresh the group scorecards options.

Note: The Number of Refresh Threads option is not supported when partitioning is enabled.

Partitioning is available to speed the processing time for identity refresh tasks and level the load on the machines running these tasks. Partitioning is used to break operations into multiple pieces, or partitions. Each partition is then placed in a global queue, and machines, or hosts, in a cluster compete to execute the partitions in the queue. Machines are added or removed from the cluster dynamically with automatic balancing. If a machine fails or is taken down while processing a partition, the partition is placed back into the queue and reassigned to a different machine. A single result object is shared by all partitions and is continually updated so you can monitor the overall progress of the partitioned operation. When all partitions have finished executing the result is marked complete. See, “Partitioning” on page 63.

The information scanned and updated is determined by the following criteria when the task is created or edited. You can use any combination of options to build a task.

Table 28—Identity Refresh Options

Option	Description
Optional filter string to constrain the identities refreshed	A filtering string used to limit the number of identity cubes updated by this task. For example you can limit the refresh to one department within your enterprise, such as Finance, by entering: department == "Finance"
Optional list of group or population names to constrain the identities refreshed	A filtering string used to limit the number of identity cubes updated by this task. For example you can limit the refresh to one group or population within your enterprise.
Refresh identities whose last refresh date is before this date	Refresh any identities not refreshed since the date entered. Enter and date manually or click the **"..."** icon to display the calendar view. Use this to recover from a refresh that ended abnormally. For example, you start a refresh task and it runs for a day before stopping abnormally. After resolving the issue with the task, instead of repeating the refresh of all the identities that completed before the task stopped, you can only refresh the ones that were missed on the last refresh. Enter the approximate date the last refresh stopped and only refresh the remainder.
Refresh identities whose last refresh date is at least this number of hours ago	Enter the number of hours manually. Use this option to refresh identities that have not been refreshed recently. The time is in this option is relative rather than absolute. Instead of remembering a specific task launch date and typing that in each time you run the refresh task you can have just one task and run that repeatedly. For example you can run it for every thing more than an hour old.
Refresh identities whose last refresh date is within this number of hours	Enter the number of hours manually. Use this option to refresh identities that were refreshed recently. The primary use case for this is to refresh things that were recently touched by aggregation. For example, if you have several aggregation sources but those sources tend to touch different subsets of all identities, and you would rather not refresh the identities that were not touch be the last aggregation.

Table 28—Identity Refresh Options

Option	Description
Include modified identities in the refresh window	Refresh any identities modified within the specified time frames. There are two dates stored on each Identity, the date of last refresh and the date of last modification. The last refresh date is set whenever you run the refresh or aggregation tasks and the identity is changed in some way. The last modification date is set whenever you edit the identity in some way outside of a refresh or aggregation task, for example from a Lifecycle Manager workflow or a custom task. Use this option to refresh identities that were edited within a period of time, but not necessarily by the refresh task. For example, you might do a full refresh once a week but during the week people were adding or removing roles, changing extended identity attributes, doing manual correlation, or changing identities in some other way. Most of those cases have options to do a targeted refresh immediately after the change happens but this is not always the case and sometimes it is better to batch up a number of refreshes rather than have hundreds of individual refreshes occurring concurrently. If you ran the refresh task with one of the date-based options you would not necessarily pick up identities that were manually edited. If you want to include those select this option.
Refresh only identities marked as needing refresh during aggregation	Only refresh identities marked as needing refresh during the most recent aggregation task.
Do not reset the needing refresh marker after refresh	Do not clear the needing refresh marker set during aggregation. Use this option if you have multiple refresh tasks scheduled, such as entitlement and risk refresh. Then you can set the final refresh to clear the markers.
Exclude identities marked inactive	Exclude inactive identities from the refresh.
Refresh identity attributes	Update identity cubes with any changes made to the attributes used to define identities.
Refresh Identity Entitlements for all links	Refresh any account attribute mark as an entitlement in the application schema. This process is resource intensive as it refreshes all entitlement values for all links.
Refresh manager status	Update all identity cubes in which the manager status has changed. For example, if a user was promoted to manager in their department, their identity cube would be updated by this task.
Refresh assigned and detected roles and promote additional entitlements	Update any assigned or detected role assignments that have change since the last time this task was run. Any additional entitlements found in this refresh are promoted during this task.

Table 28—Identity Refresh Options

Option	Description
Provision assignments	Provision any assigned roles and entitlements detected since the last time this task was run.
Disable deprovisioning of deassigned roles	Prevents assigned roles from being deprovisioned after they have been deassigned.
Refresh role metadata for each identity	Update information about the identity's relationship to their role. For example, information regarding whether or not an identity has all the roles required by the given role. Note: This option must be selected in order to generate Role Statistics.
Enable manual account selection	Sent Account Selection Notification emails to users with more than one account on any application where the system cannot determine the provisioning account. By default, no provisioning is done in this case.
Synchronize Attributes	Provision identity mapping targets if their value has changed.
Refresh the identity risk scorecards	Update Identity Risk Scores with any information discovered by the scan performed by this task.
Maintain identity histories	Update the identity history by creating a snapshot of any identities with information that has changed since the last refresh.
Refresh the group scorecards	Update Group Risk Scores with any information discovered by the scan performed by this task. Partitioning is disabled if you select this option.
Clean up groups definitions that are no longer referenced	Delete un-referenced group definitions. This option is only supported if it is selected in conjunction with the Refresh the group score card option and they run in the same task.
Check active policies	Scan for active policies and apply those policies to the identities included in the task.
Keep previous violations	Maintain a history of violations that are no longer active.
A comma separated list of specific policy names. When set this overrides the default policies	Scan for and apply only those policies included in this list to the identities included in the task.
Refresh assigned scope	Refresh assigned scope based on changes discovered.
Disable auto creation of scopes	Do not automatically assign scope to identities as part of this task.

Table 28—Identity Refresh Options

Option	Description
Mark dormant scopes after refresh	Mark scopes that are not assigned to any identities as dormant. Partitioning is disabled if you select this option.
Process Events	Enable event certifications. Use the snapshots created during aggregation to approximate the previous state of the identities at the beginning of the refresh. This copied identity is compared to the updated identity to determine if event certifications are launched.
Refresh logical application links	Scan for changes to composite applications and refresh the link information.
Promote managed attributes	When enabled, any values for entitlement or permissions encountered while running the task automatically get promoted as managed attributes
Number of Refresh Threads	Specify the number of concurrent threads used during task processing. The number of threads should not exceed 10. This option is not supported whit partitioning enabled.
Always launch the workflow (even if the usual triggers don't apply)	Launch a workflow for each identity even if no identity triggers or provisioning policy questions apply.
Enable the generation of work items for unmanaged parts of the provisioning plan	Create work items for role entitlements that are not managed by available connectors or provisioning integration modules so the appropriate action can be taken.
Disable connector lookup of managers that do not correlate	Disable the default MANAGER_LOOKUP feature and stop the automatic lookup/bootstrap of the manager account at the connector level.
Enable partitioning	Enable partitioning of this task across multiple hosts. Partitioning has to be configured before this option is valid.

Identity Request Maintenance

The Identity Request Maintenance task scans all Lifecycle Manager access requests to ensure that all identity change requests were provisioned.

Table 29—Identity Request Maintenance Options

Option	Description
Max age (in days) for Identity Request objects	The maximum number of days that an identity request object (AccessRequest) is stored in the IdentityIQ database before it is removed. Set this according to your policy on how long access request details are required. The default is zero (0), which indicates that they are stored forever.
Verify provisioning for requests?	Scan for provisioning requests which have been verified.
Number of days to attempt to verify the request with the Identity model before failing.	The number of days the task attempts to scan for verified access requests before reporting a failure. When a timeout occurs, any item not verified is left in the non-finished provisioning state, either Committed or Pending, and the overall request is marked Partially Complete if any item succeeded. If no item succeed the entire request is marked failed. Set this value based on the type of connectors and their expected provisioning times. The default setting is continuous checking forever.

Missing Managed Entitlements Scan

Missing Managed Entitlement Scan tasks scan the selected application and create any entitlement objects for items added after the application was last aggregated.

Select the applications to include in the scan. At least one application must be specified. Click the arrow to the right of the suggestion field to display a list of all applications, or enter a few letters in the field to display a list of applications that start with that letter string.

This task returns a list of entitlement names, values, and the application on which they were detected.

Novell Application Creator

Run the Novell Application Creator task to inspect Novell IDM applications and retrieve information about the applications to which they are connected. This task generates an IdentityIQ application for applications connected to the Novell IDM application specified. The applications generated by this task are added to the list of applications in IdentityIQ.

Table 30—Novell Application Creator Options

Option	Description
Novell IDM Application	Select a Novell IDM application to inspect and from which applications should be generated.
Novell Connected Applications List	Specify the connected applications for which application should be created in IdentityIQ.

OIM Application Creator

Run the OIM Application Creator task to inspect Oracle Identity Manager applications and retrieve information about the applications to which they are connected. This task generates an IdentityIQ application for applications connected to the OIM application specified. The applications generated by this task are added to the list of applications in IdentityIQ.

Table 31—Novell Application Creator Options

Option	Description
OIM Application	Select an OIM application to inspect and from which applications should be generated.

Policy Scan

The Policy task type is used to run policies against identity cubes and update identity score cards with any policy violations discovered. IdentityIQ provides the Check Active Policies task as a global policy update task.

The information scanned and updated is determined by the following criteria when the task is created or edited. You can use any combination of options to build a task.

Table 32—Policy Scan Options

Option	Description
Optional filter string to constrain the identities refreshed	A filtering string used to limit the number of identity cubes updated by this task. For example you can limit the refresh to one department within your enterprise, such as Finance, by entering: department == “Finance”
Optional list of group or population names to constrain the identities refreshed	Use this list to further limit the number of identities included in this policy scan.
Apply all active policies	Scan for active policies and apply those policies to the identities included in the task.
A comma separated list of specific policy names. When set, this overrides the default policies	Scan for and apply only those policies included in this list to the identities included in the task.

Propagate Role Changes

Note: IdentityIQ does not propagate role changes for entitlements on applications that do not support direct provisioning and would require the creation of multiple work items. If required, a business process can be enabled in the System Configuration settings to handle that situation.

The Propagate Role Changes task updates any identities that have assigned roles whose associated entitlements have changed. This is the only task that can propagate the removal of entitlements from an assigned role.

Table 33—Propagate Role Changes Options

Option	Description
Number of minutes task should run	The number of minutes for the task to run. The task stops only after finishing current event processing.
Check active policies	Scan for active policies and apply those to the identities included in the task.
Keep previous violations	Mark old policies as inactive but do not delete them.
A comma separated list of specific policy names. When set, this overrides the default policies.	Scan for and apply only those policies included in this list to the identities included in this task.
Enable Partition	Allow the task to split into partitions and run across multiple threads and hosts, if available.

- Once you have completed customizing your task options, click Save for later use or Save and Execute to save the task and run it immediately.
- After executing the task, the Task Result page displays the following output:
- **Number of Identity Updates**: displays the total number of Identity updates propagated. It is different than number of Identities updated, since multiple role events include some common identities and are counted multiple times, for each role event.
- **Number of Events Processed**: displays the total number of role events propagated. This is not the number of role modifications but the role change events in the queue. As single role modification results in multiple role change events in the queue.
- **Number of Events Pending:** displays the total number of pending role change events in the queue. If timeout is not defined, Role Propagation task completes only after propagating all the events. If timeout is defined, there could be pending events in the queue even after successful completion of this task.
- **Number of Events with No Impacted Identities:** displays the total number of events which are not impacting on connected identities. This event count is based on those bundles which are not directly assigned to the identities.
- Role change events are propagated sequentially and are not consolidated to cancel out redundant changes.
- If Refresh Identity Task is run before Role propagation task, and if it adds any entitlement as part of role changes, processing of role change event through Role Propagation Task would be redundant.
- In case of retry status, the transaction would be marked as failed and role propagation task would be

stopped.

- While adding an entitlement, if account is missing, transaction would be marked as failed and role propagation task would be stopped. User has to run Refresh Identity task to resolve this.
- While processing an event, if the following exception is from target system, the task would remain blocked until the events are successfully processed.
- `mandatory group cannot be removed`
- This issue can be resolved by deleting the event from the database.
    - When Role Propagation task is under execution, if user creates events in database, these events would not be considered by the current task. These events would be considered for the next task execution.

Refresh Logical Accounts

The Refresh Logical Accounts task type is used to refresh composite accounts for all identities that could, potentially, have a logical account on the applications selected. This refresh occurs without performing aggregation on the logical or tiered applications containing the links.

A logical account rule is run on each identity that has a logical link, or a link on the primary tier application of the logical application. If no primary tier has been defined, the rule is run on all identities that have an account on any of the tier applications.

The information scanned and updated is determined by the following criteria when the task is created or edited. You can use any combination of options to build a task.

Table 34—Refresh Logical Accounts Options

Option	Description
Logical Applications	Select composite applications to refresh from the drop down-list.
Refresh identities whose last refresh date is before this date	Refresh any identities that have not been refreshed since the date entered. Enter and date manually or click the **“...”** icon to display the calendar view.
Refresh all application account attributes	Perform an aggregation of identity information on each application and update the account attributes on each identity as required. Selecting this attribute initiates a full application aggregation for each identity included in this task. This might impact the performance of IdentityIQ.
Refresh identity attributes	Update identity cubes with any changes made to the attributes used to define identities.
Refresh manager status	Update all identity cubes in which the manager status has changed. For example, if a user was promoted to manager in their department, their identity cube would be updated by this task.
Refresh the identity risk scorecards	Update Identity Risk Scores with any information discovered by the scan performed by this task.
Maintain identity histories	Update the identity history by creating a snapshot of any identities with information that has changed since the last refresh.

Table 34—Refresh Logical Accounts Options

Option	Description
Refresh the group scorecards	Update Group Risk Scores with any information discovered by the scan performed by this task.
Apply all active policies	Scan for active policies and apply those policies to the identities included in the task.
A comma separated list of specific policy names. When set this overrides the default policies	Scan for and apply only those policies included in this list to the identities included in the task.
Number of Refresh Threads	Number of threads to use simultaneously while running this task.

Role Index Refresh

A role index refresh task updates all role information and creates the indexes needed to perform role searches. You must run this task before performing any role searching.

Run Rule

A task used to run an arbitrary rule with a series of name/value pairs.

You must have to configure some return statement as string. From your code, you have to return some meaningful string to the task. In your task definition declare:

```
         <Returns>
          <Argument name="tskSuccess" type="string">
            <Prompt>Task  Result:</Prompt>
          </Argument>
        </Returns>
```

And in your code:

```
String tskSuccess = "failed";
if ( Do some condition check here) {
//Do something;
String tskSuccess = "Success";
}
return tskSuccess;
```

The rule is expected to return a string value representing its status. Any string other than `Success` results in a failed task result.

Sequential Task Launcher

A sequential task launcher initiates the specified tasks in the order defined. This enables you to run tasks sequentially without having to schedule each separately based on estimated run times.

Table 35—Sequential Task Launcher Options

Option	Description
Enter the list of tasks you would like to run. Tasks are run in the order that they are entered.	Select the tasks you would like to run and the order in which they should run.
Print log statements to indicate which tasks have been completed.	Select to print log statements so the sequential tasks can be tracked.
Cease execution if one of the executing tasks encounters an error.	Select to stop the sequential task if one of the tasks in the list fails. If this option is not selected the task continues in order.

System Maintenance

SailPoint provides System Maintenance tasks with the IdentityIQ application, the Work Item Expiration Scanner, Mitigation Expiration Scanner, System Maintenance, System Maintenance Object Pruner, Role Overlap Analysis, and the Synchronize Roles task. These tasks are configured, by default, to run in the background of the application and update score card, application, and role information as needed.

The Work Item Expiration Scanner checks for work items that were assigned but have not been completed by the set expiration date.

The Mitigation Expiration Scanner checks for roles or entitlements for which the exceptions allowed during certification have expired.

The Perform Maintenance task prunes identity snapshots, task results, access request attachments, and certifications, escalates orphaned work items, and performs other background maintenance tasks.

The System Maintenance Object Pruner prunes objects in batches to improve performance. This task is not part of the System Maintenance task pruning operations and is run independently when necessary. This task is always run with partitioning enabled.

The Role Overlap Analysis performs impact analysis on a specified role. The task result name is annotated with the name of the selected role so you can tell multiple analysis results apart.

The Synchronize Roles task synchronizes IdentityIQ roles with the roles on the identity management systems that are configured to work through a provisioning provider.

Target Aggregation

A target aggregation task scans applications and aggregates activity targets from those applications. These targets are then used for activity monitoring and risk assessment.

Select the applications to include in the scan. If no applications are specified, all applications are included. Click the arrow to the right of the suggestion field to display a list of all target sources.

Select **Include empty targets** to aggregate targets with no associated users or groups. By default, empty containers will not be included.

How to Complete Task Work Items

Task work items are generated by task that require sign off on the results they create, and those sign off request that are forwarded by a designated signer. Sign off request are displayed on your Home Page and you are notified through an email when the work item is created.

Sign off decisions are retained with the task results for tracking purposes.

1. Click on a sign off type work item to display the sign off request.
2. Review the work item information in the Summary section.
3. Review the Comments section for any information associated with this work item.
   Refresh
4. Click **Click to View Task Results** in the Details sections to display the Task Results page.
5. Click **Return to Work Item** when you have completed your review of the task results.
6. Click on an action button to open the associated comments dialog and conclude this work session.

   If you sign off on, or reject the sign off request, the status of the task results is updated to reflect that decision. You must specify a recipient If you forward the work item.

Chapter 6: Alerts

Alerts are created using IdentityIQ File Access Manager (FAM) based on activity data - actions users take on resources that are part of an application that FAM is monitoring. FAM can be configured to create alerts when the user action is considered unexpected, potentially risky, or inappropriate. It is possible to configure alerts for any behavior. You can choose to use this functionality more broadly (e.g. for non-risky or non-problematic activities that someone wants to use as a process trigger).

This integration additionally enables you to trigger actions in IdentityIQ in response to an alert. Specifically, alerts aggregated into IdentityIQ can be used to drive three different response actions. A single alert can trigger more than one response action:

- Launch a certification
- Launch a workflow
- Send an email notification

Alerts Page

Use the Alerts page to view existing alerts for your enterprise. To limit the number of alerts displayed in the table, use the filtering options.

Table 36—Filter Alerts Definitions

Column Name	Description
Name	The name of the alert.
Source	Application associated with the alert.
Native Id	Native identifier of the application with which the alert is associated.
Type	Alert type.
Target Type	Type of the object that triggered the alert.
Target Name	Name of the object that triggered the alert.
Alert Date Start	Date and time at which this alert was triggered.
Alert Date End	Date and time at which this alert expires.
Last Processed Start	Last date and time this alert was triggered.
Last Processed End	Last date and time this alert process finished.
Acted Upon	Select True if this alert matched an alert definition and an alert action was triggered.

Alert Definitions Page

Use the Alert Definition page to view a list of existing alert definitions. The Alerts page contains the following information:

Table 37—Alert Definitions Page Field Description

Field Name	Description
Name	A descriptive name of this alert. This is the name that displays on the Alerts page.
Display Name	Label that is displayed on the alert.
Description	A brief description of the alert definition.
Created	The date and time the alert definition was created.
Status	The status of the alert definition: Active – the alert definition is currently being used Inactive – the alert is not being used

Create Alert Definition

The Create Alert Definition page contains the following information:

Table 38—Create Alert Definition

Field Name	Description
Name	A descriptive name of this alert. This is the name that displays on the Alerts page.
Display Name	Label that is displayed on the alert.
Description	A brief description of the alert.
Owner	The alert owner, not necessarily the identity who triggered the alert.
Match Rule	Enables more complex matching logic.
+Add	Option to add a **Match Term**.
Source	Application name that triggers the alert.
Attribute	The display name of an account attribute derived from the attribute and its associated application.
Value	Value for the selected attribute that will trigger an alert during alert processing.
Action Type	Action to be taken when the alert is created. This can either be a notification, certification, or a workflow, or a combination of the available actions.
Email Template	Template used for the notification email. If none is selected, a system default is used.
Email Recipients	List of users to receive the alert notification.

How to Create an Alert Definition

Alerts are created using the Alert Definitions tab. Use this procedure to create new alert.

Procedure

1. Click the **Alert Definitions** tab on the Alerts page.
2. Click **+New**.
3. Enter the alert information.

See "Create Alert Definition" on page 106 for detail description of the Alert Definition page.

4. Click **Save** to save the alert and return to the Alerts page.

Edit Alert Definitions

Use the Edit Alert Definition to edit existing rules. The Edit Alert Definitions page contains the following information:

Field Name	Description
Name	A descriptive name of this alert. This is the name that displays on the Alerts page.
Display Name	Label that is displayed on the alert.
Description	A brief description of the alert.
Owner	The alert owner, not necessarily the identity who triggered the alert.
Source	Application name that triggers the alert.
Attribute	The display name of an account attribute derived from the attribute and its associated application.
Value	Value for the selected attribute that will trigger an alert during alert processing.
Match Rule	Enables more complex matching logic.
Action Type	Action to be taken when the alert is created. This can either be a notification, certification, or a workflow, or a combination of the available actions.
Workflow	Defines the workflow structure and steps involved in the workflow processing.
Email Template	Template used for the notification email. If none is selected, a system default is used.
Email Recipients	List of users to receive the alert notification.

How to Edit an Alert Definition

Alerts are edited using the Alert Definitions tab. Use this procedure to edit existing alerts.

Procedure

1. Click the **Alert Definitions** tab on the Alerts page.
2. Select an alert and lick **Edit** in the Actions column.
3. Enter the alert information.

See "Edit Alert Definitions" on page 107 for detail description of the Edit Alert Definitions page.

4. Click **Save** to save the alert and return to the Alerts page.

How to Filter Alerts

Use the filtering options to limit the number of alerts displayed in the table. You can filter by any of the 11 fields. Use this procedure to filter through existing alerts.

Procedure

1. Click the **Alert** tab on the Alerts page.
2. Click **Filter**.
3. Enter filtering information.
4. Click **Apply** to save the filter options.

Chapter 7: Work Items

The Work Items page provides a central location where you can view and manage work items that are assigned to you or to a workgroup of which you are a member. A work item is anything that requires an action before it is completed. Work items can be entire processes, such as access reviews, or any piece of a process, such as the approval of one entitlement for one user on one application.

Work Items on the Home Page

When a work item is created and you have a Notifications card on your Home page, the Notifications card displays the number of work items assigned to you. To access the work items, click the card or go to the navigation bar and click **My Work -> Work Items**.

To manage work items, refer to the following:

- "Work Item Administration" on page 109
- "Work Item Archive" on page 110

Work Item Administration

Note: To edit priorities in IdentityIQ, the "Allow priority editing on work items" setting must be selected on the Work Item tab IdentityIQ Configuration page located under the gear icon.

If a work item is created for a user who is no longer active in IdentityIQ, it is forwarded to the manager or supervisor for that user. If no manager is listed, the work item is assigned to the IdentityIQ administrator. Use escalation rules to determine the proper escalation path for orphaned work items. Escalation rules are created and set during the configuration and implementation of the product. Orphaned work items are discovered and identified during the Perform Maintenance task.

Use **Sort By** to customize the sort order of the work item list, or use the newest to oldest icon to flip the list.

Use **Filter** to limit the number of work items displayed, or search on a specific work item using the search field.

Click **View Archive** to see a list of completed work items.

Use the **Show...** drop-down list to select the work items you would like to see.

The Work Items page displays the following information:

Table 39—Work Item Page

Column Name	Description
Priority	Specifies the priority level to which the work item was designated. Use the drop-down list and edit the priority level. This edit is visible in the Work Items Manager and Inbox of the identity to whom the work item is assigned, as well the outbox of the person that assigned the work item.
Type	The type of work item.
Name	The name of the work item.
Created	The date the work item was assigned.
ID	Identification number assigned to the work item.

Table 39—Work Item Page

Column Name	Description
Owner	The name of the identity who has purview over the work item.
Requestor	The name of the user who assigned this work item to you.

Click the information icon to see the Details dialog containing work item and identity details, as well as any forwarding history associated to the work item.

Note: Work items can only be assigned if the assignee of the work item is a member of the same workgroup as the person who is assigning the work item.

Click the forward icon to open the **Forward Work Item** dialog.

The Manage Work Items table includes the following types of work items:

Work Item Archive

Use the Work Item Archive page to view completed work items. Only work items types that are configured in System Setup can be viewed on the Work Item Archive page. To access the system settings for Work Items, navigate to the IdentityIQ **Configuration > Work Items** tab under the gear icon.

Click the drop-down list to specify if your table displays all work items assigned to you and any groups to which you belong, only your own, personal work items or only the work items assigned to a selected workgroup.

To customize the information displayed in the Work Item Archive table, mouse over one of the header rows, click the drop-down arrow to reveal the sub-menu and select the desired columns from the Columns pop-out menu.

Click a line item launch the View Work Item page which displays detailed information about the work item.

Table 40—Work Item Archive Column Descriptions

Column Name	Description
ID	Identification number assigned to the work item.
Name	The name of the work item.
Type	The type of work item.
Requestor	The name of the user who assigned this work item to you.
Workgroup	Displays the workgroup to which this work item is assigned if applicable.
Owner	The name of the identity who has purview over the work item.
Completed By	The date the work item was completed
Created	The date the work item was assigned.
Modified	The date changes, if any, were made to the work item.
Archived	The date the work item was archived.
Priority	Specifies the priority level to which the work item was designated. Use the drop-down list and edit the priority level. This edit is visible in the Work Items Manager and Inbox of the identity to whom the work item is assigned, as well the outbox of the person that assigned the work item.
Access Request ID	Identification number designated for the Lifecycle Manager access request.

Chapter 8: IdentityIQ Console

The IdentityIQ Console is the command line utility for interfacing with IdentityIQ. This document lists the console commands and their descriptions.

Launching the Console

The IdentityIQ console requires the System Administrator capability.

By default, the console tries to authenticate with the default user/password `spadmin/admin`. If authentication fails, you are prompted for a user name and password. Specify the user name and password on the command line.

For example:

```
iiq console -u amy.cox -p mypassword
```

The console prompts for user input if the password is omitted, and will not launch if the credentials supplied are not associated with an identity that has console access.

Note: Authentication is disabled if there are no identities. This case is encountered during IdentityIQ setup, before init.xml is imported.

The IdentityIQ Console (iiq console) is launched by executing the `iiq.bat` file found in the *`Installation Directory`*`/WEB-INF/bin` directory. From a command prompt, launch the console with the command as shown for each operating system type:

Table 41—Console Launch Commands

Operating System	Command
Windows	`iiq console`
Unix	`./iiq console -j`

Note: The -j option turns on the JLine Java library for handling console input, enabling some ease-of-use functions such as command history recall. Command history recall is enabled in Windows without this library, so this parameter is not required in the Windows environment.

The > prompt character indicates that the console is running and ready to accept commands.

Viewing the List of Commands

The **help** command displays a list of all commands available in the console along with a short description of each. At the command prompt, enter `help` or? to see this full list of available commands.

Table 42—List of Console Commands

Command	Description
?	Display command help

Table 42—List of Console Commands

Command	Description
help	Display command help
echo	Display a line of text
quit	Quit the shell (same as exit)
exit	Exit the shell (same as quit)
source	Execute a file of commands
properties	Display system properties
time	Show how much time a command takes to run
xtimes	Run a command x times
about	Show application configuration information
threads	Show active threads
logConfig	Reload log4j configuration
Objects	
dtd	Create dtd
summary	Summarize objects
classes	List available classes
list	List objects
count	Count objects
get	View an object
checkout	Checkout an object to a file
checkin	Checkin an object from a file
delete	Delete an object
rollback	Rollback to a previous version
rename	Rename an object
import	Import objects from a file
importManagedAttributes	Import managed attribute definitions from a CSV file
export	Export objects to a file
exportManagedAttributes	Export managed attribute definitions to a CSV file
exportJasper	Exports only the jasperReport xml contained in a JasperTemplate object
associations	Show target associations for an object
Identities	
identities	List identities
snapshot	Create an identity snapshot
score	Refresh compliance scores

Table 42—List of Console Commands

Command	Description
listLocks	List all class locks
breakLocks	Break all class locks
Tasks	
tasks	Display scheduled tasks
run	Launch a background task
runTaskWithArguments	Launch a task synchronously with arguments
terminate	Terminate a background task
terminateOrphans	Detect and terminate orphaned tasks
restart	Restart a failed task if possible
send command	Send an out-of-band task command
taskProfile	Display task profiling report
Certifications	
certify	Generate an access certification report
cancelCertify	Cancel an access certification report
archiveCertification	Archive and delete an access certification report
decompressCertification	Decompress an access certification archive
Groups	
refreshFactories	Refresh group factories (but not groups)
refreshGroups	Refresh groups (but not factories)
showGroup	Show identities in a group
Workflow	
workflow	Start a generic workflow
validate	Validate workflow definition
workItem	Describe a work item
approve	Approve a work item
reject	Reject a work item
wftest	Run the workflow test harness
Tests	
rule	Run a rule
parse	Parse an XML file
warp	Parse an XML object and print the re-serialization
notify	Send an email
authenticate	Test authentication

Table 42—List of Console Commands

Command	Description
authenticateWithOptions	Test authentication with options
simulateHistory	Simulate trend history
search	Run a simple query
textsearch	Run a full text search
certificationPhase	Transition a certification into a new phase
impact	Perform impact analysis
event	Schedule an identity event
expire	Immediately expire a workitem that has an expiration configured. If the workitem is type Event it'll also push the event forward with the workflower
connectorDebug	Call one of the exposed connector methods using the specified application
encrypt	Encrypt a string.
sql	Execute a SQL statement
hql	Execute a search based on a Hibernate Query Language statement.
updateHql	Update the hql search.
date	Displays the current system date/time and its UTIME (universal time) value (Optional UTIME parameter causes the command to display the date/time corresponding to the provided UTIME value.)
shell	Escapes out to the command line and run the command specified.
meter	Toggles metering on and off; while metering is on, the console reports some timing statistics for each command executed. Meter information is displayed after the results of each command as it is executed.
compress	Compress the contents of a file to a string that can be included within an XML element.
uncompress	Return a compressed, Base64-encoded file to its uncompressed format.
clearEmailQueue	Remove any queued emails that have not been sent
provision	Evaluate a provisioning plan
lock	Lock an object
unlock	Break a lock on an object
showLock	Show lock details
clearCache	Clear the object cache
service	Service management
oconfig	Analyze ObjectConfigs
plugin	Install and manage plugins
recommender	Manage and test recommendations

Command-Line Parameters

You can use parameters with the iiq console commands to manage command behavior.

Table 43— List of Command-Line Parameters

Command	Description
-u <username> -p <password>	Run the console using the supplied username and password for authentication. Example: `-u mary.johnson -p mypwd` If you provide a username but omit the password, the iiq console prompts for the password value.
-j	Used in Unix systems only. Adds improved command editing and history support. Use cursor-up and cursor-down to navigate console command history.
-h <hostname>	Override the hostname used for the console. Example: `-h consoleA`
-c <command>	Run the given command, and then exit. Example: `-c "list Certification"`
-f <filename>	Run the commands read from the provided file, then exit. Example: `-f myCommands.txt`
-e <CSV of services to start>	Automatically start the services specified in the provided comma-separated list. Example: `-e Heartbeat,Task,Reanimator`
-heartbeat	Force the Heartbeat service to automatically start. This is the equivalent to `-e Heartbeat`

Command Syntax

The syntax for any console command that requires parameters can be determined by entering that command with no arguments.

```
> workflow
usage: workflow name [varfile]
```

Note: Command names are case sensitive and must be entered as shown in the command list. Parameters are not case sensitive.

Some commands take no arguments and execute if entered. This table contains a list of the commands that require no arguments.

Table 44—Console Commands That Do Not Require Arguments

Command	Action
? or **help**	Lists all available console commands
quit or **exit**	Exits the console shell
classes	Lists all classes
refreshGroups	Refresh group indexes (Optional group name or ID can be specified)
refreshFactories	Refresh set of GroupDefinitions for a GroupFactory (Optional factory name or ID can be specified)
logConfig	Reloads log4j configuration from log4j2.properties file
summary	Lists all classes and the count of objects of that class in the system
properties	Displays Java properties of the server where IdentityIQ is installed
about	Displays application configuration information
threads	Shows a list of active threads
tasks	Writes a list of all currently scheduled tasks, in a columnar layout, to the console (stdout)
identities	Writes the Name, Manager, Roles, and Links for each Identity in the system to the console (stdout)
date	Displays the current system date/time and its UTIME (universal time) value (Optional UTIME parameter causes the command to display the date/time corresponding to the provided UTIME value.)
status	Reports current running status of the task and request schedulers
meter	Toggles metering on and off; while metering is on, the console reports some timing statistics for each command executed. Meter information is displayed after the results of each command as it is executed.
clearEmailQueue	Deletes all queued but unsent email messages
clearCache	Clears the IdentityIQ object cache

Syntax for Redirecting Command Output

Most of the commands report data or error messages to the console or standard out (stdout) for the system. The output for any command can be redirected to a file by specifying > *`filename`* at the end of the command.

This example redirects the output from the **get** command to a file:

```
> get identity Adam.Kennedy > c:\output\AdamKennedyID.xml
```

Console Commands

In this document, the list of console commands is subdivided based on how frequently they are likely to be used in a production environment.

- **Commonly Used Commands** — commands that system administrators should know well and use frequently.
- **Less Commonly Used Commands** — commands that might need to be run periodically, such as when working with SailPoint Support to resolve an issue, but are not used regularly.
- **Seldom Used Commands** — developer commands that are rarely useful in a production environment.

Commonly Used Commands

This section lists and documents the syntax and actions of the most commonly used console commands.

Help and ?

These two commands list all the available console commands.

Syntax	`?` `help`

Examples	`> ?`
	
	`> help`

Result	Lists all commands available in the console

Exit and Quit

These two commands exit the console shell, returning the user to the operating system command prompt.

Syntax	`exit` `quit`

Examples	`> exit`
	
	`> quit`

Result	Exits console shell and returns user to the operating system command prompt

Source

The **source** command runs commands from a script file. The commands on each line in the file are executed by the console sequentially.

Syntax	`source` *`filename`*

Examples	`source c:\data\cmdfile.txt`
Result	Runs the console commands in the `c:\data\cmdfile.txt` file sequentially

List

The **list** command lists all objects of the specified class, constrained by any specified filter. If this command is specified without arguments, the command syntax is displayed, followed by a list of all available classes whose objects can be listed. This is helpful in locating objects within the system and in identifying object names to use as parameters on other commands.

Syntax	`list` *`classname`* `[`*`filter`*`]`
	`filter: xxx - names beginning with xxx`
	`xxx* - names beginning with xxx`
	`*xxx - names ending with xxx`
	`*xxx* - names containing xxx`

Examples	`> list application ent*`
Result	Lists all application objects whose names begin with ent

Get

The **get** command displays the XML representation of the named object.

Syntax	`get` *`classname <objectname`*`re or` *`ID>`*
Examples	`> get identity Adam.Kennedy`
Result	Displays the Adam.Kennedy Identity in XML format

Note: This command only displays the object to the console (stdout), it does not export the object. The output can be redirected to a file if the user has write access to the server's file system.

```
> get identity Adam.Kennedy > c:\output\AdamKennedyID.xml
```

Other alternatives for getting the XML representation of an object into a text file include:

- copying and pasting contents of this command's stdout into a text file
- retrieving the object's XML from the IdentityIQ Debug pages
- using the **checkout** command (described next) to write the XML representation of an object to a text file

Checkout

The **checkout** command writes a copy of the XML representation of the requested object to the specified filename. The file can be used for review or for moving objects from one environment to another, for example, from the user acceptance testing environment to production. Organizations doing custom development on rules, workflows, etc. might use **checkout** to extract any of these objects to a file for modification.

Syntax	`checkout` *`class name`* `<`*`objectname`* `or` *`ID`*`>` *`file`* `[-clean [=id,created…]]`
Examples	`> checkout rule “Cert Signoff Approver” certrule.xml`
Result	Writes a copy of the Cert Signoff Approver rule's XML representation to the file `certrule.xml`

The **-clean** option can be used to remove all values that do not transfer between IdentityIQ instances, such as created and modified dates as well as globally unique ID values (GUIDs). Specifying the **-clean** option with no qualifiers cleans the id, created, modified, and lastRefresh attributes. The **-clean** option can also be used to explicitly clear specific fields by name. The fields to clear must be listed in a comma separated list.

Checkin

The **checkin** command reads a file containing an object's XML representation and saves the object into the database. If the object is a workItem, the command invokes the workflower to process the workItem. If the object is a bundle (role) and the approve parameter is specified, a role approval workflow is launched. For all other object types, and for bundles that are submitted without the approve parameter, the object is saved into the database.

Note: The command's syntax parsing allows the approve parameter to be specified for any object but it only impacts the processing on Bundle objects.

Syntax	`checkin` *`filename`* `[approve]`

Examples	`> checkin newRole.xml approve`
	
	`> checkin bobSmithID.xml`

Result	First example saves Identity Bob Smith, as represented by the XML in bobSmithID.xml, into to the database; overwrites existing or adds new record Second example launches an approval workflow for the bundle object represented by the XML in newRole.xml

Note: If an Import file is specified as the input file for this command, only the first object in the file is checked in; the rest are ignored and a warning message is displayed to the console (stdout).

Delete

Note: This action cannot be undone and should be used with extreme caution and only in rare circumstances.

The **delete** command deletes the named object and removes all of its owned, or subordinate, objects. In a production environment, this is not recommended unless specifically directed by SailPoint Support.

Syntax	`delete` *`classname`* `<`*`objectname`* `or` *`ID`*`>`
Examples	`> delete identity bob.smith`

Result	Removes Identity Bob Smith and all of his associated objects from the system

Note: Wildcards can be used on the <*object name* or *ID*> argument:
— * - all objects of the specified class (use with extreme caution!)
— xxx - all objects whose name or ID contains xxx

Import

The **import** command imports objects into IdentityIQ from an XML file. This command can be used on a file that contains a Jasper report, a SailPoint import file, or an object of one of the standard object classes. The file contents are evaluated and processed based on the first tag in the file:

- *JasperReport*: Jasper report
- SailPoint: SailPoint import object; can contain multiple regular objects in one file as well as an ImportAction tag that directs how the contents of the file are processed, for example, merge, include, execute, logConfig.
- Anything else: assumed to be a single regular object

Syntax	`import [-noids]` *`filename`*

Examples	`> import init.xml`
	
	`> import -noids init.xml`

Result	The first example Imports the contents of the file `init.xml` into the IdentityIQ database. In the second example, all ID attributes are removed before parsing occurs. This action is a normal part of the initialization process for IdentityIQ.

Syntax	`import [-noids][-noroleevents]` *`filename`*

Examples	`> import -noids bundles.xml`
	
	`> import -noids -noroleevents bundles.xml`

Result	The first example allows user to import the events. It removes all ID attributes before parsing. The second example disables generation of role change events for role propagation. Select 'Allow Role Propagation' option from the Global Settings -> IdentityIQ Configuration -> Roles option in UI.

This is one of the most commonly used commands. Installations who manage their workflows and rules in an external source code control system, for example, use this command to bring changes to those objects into IdentityIQ once they have been modified in their external XML representations.

Export

The **export** command writes all objects of a given class to a specified filename. This is commonly used in gathering objects from IdentityIQ to deliver to SailPoint Support as resources in resolving tickets. It is also used for moving sets of objects between environments and for managing objects outside of IdentityIQ, such as storing workflows and rules in a source code control system.

More than one class can be exported at a time to the same file by specifying all the desired class names as arguments to the command. If the **export** command is specified without any class names, all objects of all classes are exported to the specified filename.

Syntax	`export [-clean[=id,created...]]` *`filename`* `[`*`classname classname`* `…]`

Examples	`> export -clean workflows.xml workflow`
	
	`> export IdLink.xml identity link`

Result	The first example exports the entire set of workflow objects from IdentityIQ to the file `workflows.xml`, removing values from the id, created, modified, and lastRefresh attributes. The second example exports all identities and links to a single file.

ListLocks

The **listLocks** command lists all locks held on any objects of the named class. At this time, Identity is the only class for which this command operates.

BreakLocks

Note: This command should be used with caution. Locks are useful in maintaining data integrity, and breaking them at the wrong time can potentially permit conflicting updates that can result in data corruption.

Note: The unlock command can be used to break a single lock whereas this command breaks all locks held on any object in the specified class.

If a process is holding a lock but is unable to perform the required action, the lock can cause problems in other processes' performance as well. The **breakLocks** command can be used to release locks forcibly. At this time, Identity is the only class for which this command functions.

Syntax	`breakLocks` *`classname`*
Examples	`> breakLocks identity`
Result	Releases all locks held on any identity object in the system and reports to the console the identity name, lock holder, and UTIME value for the lock date/time and the lock expiration date/time.

Rule

The **rule** command runs a rule defined in the system. The rule to run is specified as a command parameter. If any input variables must be passed to the rule, they must be entered in a variable file, specified as an XML Map. The file name is then also passed as a parameter to the command.

This command can be used for testing or executing existing system rules. It can also be used to run any beanshell code snippet against the IdentityIQ database. The code is created as a rule and loaded into the system and then executed from the console. Support uses rules like this for data cleanup.

Syntax	`rule <`*`rulename`*` or `*`ID`*`> [`*`varfile`*`]`
Examples	`> rule “Check Password Policy” c:\data\pwdParams.xml`
Result	Runs the Check Password Policy rule, passing its input variables through the file `c:\data\pwdParams.xml`

Parse

The **parse** command validates an XML file. If it is in valid form and its tags match the IdentityIQ DTD, it runs successfully and no information is printed to the console (`stdout`). If errors are encountered, a runtimeException is printed to the console describing the error.

Syntax	`parse `*`filename`*
Examples	`> parse c:\data\newWorkflow.xml`
Result	Validates the XML in the file `c:\data\newWorkflow.xml` and reports any errors to the console

Less Commonly Used Commands

These commands are not frequently used in a production environment. However, it is helpful to understand and be able to use them when the need arises.

DTD

The DTD command writes the IdentityIQ DTD (Document Type Definition) to the specified file.

Syntax	`dtd `*`filename`*
Examples	`> dtd c:\DTD\IdentityIQ.dtd`
Result	Writes the IdentityIQ DTD to the file `c:\DTD\IdentityIQ.dtd`

Classes

The **classes** command lists all classes accessible from the console. These are frequently used as parameters to other commands so this list can be helpful in entering correct arguments on those commands.

Syntax	`classes`
Examples	`> classes`
Result	Lists class names for all classes accessible to the console

Count

The **count** command returns a count of the objects of the specified class.

Syntax	`count` *`classname`*
Examples	`> count identity`
Result	Displays the count of Identity objects in the system

ImportManagedAttributes

The **importManagedAttributes** command is used to set managed attribute values, including localized descriptions, through a CSV file import. This can be used to update existing managedAttributes or to create new ones.

The filename can be specified with an absolute path or can be specified relative to the current working directory. These are the specific requirements for the import file contents:

- The first line in the file must be a comment line (starting with #) that contains the column names for the data records. Column names must be specified in a comma-separated format. All column names must match managedAttribute standard or extended attributes or specify a locale/supported language.
- Subsequent comment lines can be used to specify default values for attributes that are not contained in the data records. For example, if the whole file relates to a single application, the application name could be set as a default through a single comment line.
- Blank lines are permitted in the file, and are ignored, but cannot precede the first comment line.
- The data records must consist of comma-separated data values.

  Note: Only types Entitlement and Permission are valid. Group managedAttributes are stored in IdentityIQ as a subcategory of Entitlement type managedAttributes.
- Required attributes are type, application, attribute, and value. If type is not specified in the file, type Entitlement is assumed. The others three properties must be specified in the file. Type, application, and attribute can be specified through the data columns or with a single default value in extra comment lines. The value attribute must be in the data columns and cannot have a default value.
- The data values in the columns named to match supported languages should contain the description to use for that locale.

Example File Contents:

```
# value, displayName, en_US, owner
# owner=Jeff.Wilson
# application=AD
# attribute=MemberOf
# type=Entitlement
# "CN=administrators,CN=Roles,DC=iiq,DC=com", Admins, "Administrators group",
# "CN=VPN,CN=Roles,DC=iiq,DC=com", VPN, "Remote Workers", Bob.Smith
```

Note: The test option causes the command to parse and validate the file without saving changes to the database.

Syntax	`importManagedAttributes` *`filename`* `[test]`
Examples	`> importManagedAttributes “c:\data\managedattributes.csv”`
Result	Imports managed attribute data from the file `c:\data\managedattrbutes.csv`, updating existing attributes or creating new ones from the data

ExportManagedAttributes

The **exportManagedAttributes** command exports either object properties or descriptions for managedAttributes to a CSV file. This is used to make mass changes to the managed attributes definitions or for collecting all the managed attributes in one file to review as a group.

Syntax	`exportManagedAttributes` *`filename`* `[`*`application`*`] [`*`language`*`]`
Examples	`> exportManagedAttributes “c:\data\AdamManAttrDesc.csv” ADAM en_US`
Result	Exports the managed attribute description on ADAM application to the file `c:\data\AdamManAttrDesc.csv`.

Application and language are both optional arguments and can be specified in either order. At most one application and one language can be specified at a time. If no application name is specified, managed attributes for all applications are exported. If a language is specified, only the core identifying properties of the managed attributes (type, application, attribute, value) and the descriptions for the specified locale are exported. If a language is not specified, all other object properties except descriptions are exported.

The file format generated by this command can be used in the **importManagedAttribute** command, so this command can be used to write values to a file for editing and reimporting. When an application name is specified on the export command, the application is not shown in the data rows but is specified as a default in the file header comments, as described and illustrated in the command details.

Run

The **run** command starts execution of a task that requires and accepts no arguments. Three optional parameters can be specified for this command: trace, profile, and sync.

- **Trace** — writes to the console (stdout) a trace of what happens as the task is run, depending on how the task's tracing code is written.
- **Profile** —displays the timing of certain phases of the task, the details displayed depend on the task's profile code.
- **Sync** — runs the task in synchronous execution mode, as opposed to scheduling it to run in background. If sync is specified, the control returns to the console only after the task has completed and any error messages are written to the console. If sync is not specified, the task is launched in the background and the results of the task are viewable in the taskResults object and are accessible from the console or from the user interface under **Setup** -> **Tasks** -> **Task Results**.

Syntax	`run` *`taskname`* `[trace] [profile] [sync]`
Examples	`> run “Refresh Risk Scores”`
Result	Runs the Refresh Risk Scores task in background

RunTaskWithArguments

The **runTaskWithArguments** command starts execution of a task that requires arguments. These tasks are always run in synchronous execution mode. This can only be used for tasks that accept arguments of simple data types; specifying an object as an argument is not possible here.

Syntax	`runTaskWithArguments` *`taskname`* `[arg1=val1,arg2=val2,…]`
Examples	`> runTaskWithArguments “Identity Refresh” refreshLinks=True,promoteAttributes=False`
Result	Runs the Identity Refresh task, refreshing Links for the Identities

Restart

The **restart** command restarts execution of a task that failed.

Syntax	`restart` *`taskResultName`*
Examples	`> run “Refresh Risk Scores”`
Result	Restarts the Refresh Risk Scores task in background if possible

RefreshFactories

The **refreshFactories** command can be specified with no arguments to refresh all group factories or with a specific GroupFactory name. Refreshing the group factory means identifying the group values that define each of the groups (or GroupDefinition objects). It does not refresh the list of Identities that make up each group or the statistics gathered for each group - just the list of groups themselves.

Syntax	`refreshFactories [<`*`factorname`*` or `*`ID`*`>]`
Examples	`> refreshFactories`
Result	Refreshes the GroupDefinition list associated with each GroupFactory in the system

RefreshGroups

The **refreshGroups** command refreshes the group indexes for all groups or for the specified group named as a command argument. The group indexes are collections of statistics for identities that are part of the group. The statistics include number of members, number of certifications due for certification owners in the group, number of certifications owned and completed on time by members of the group, and risk score information for members of the group. RefreshGroups only applies to GroupDefinitions that are indexed (attribute indexed=”True”).

Syntax	`refreshGroups [`*`groupname`*` or `*`ID`*`]`
Examples	`> refreshGroups`
Result	Refreshes index information for all indexed groups

ShowGroup

The **showGroup** command shows the membership (Identities) of the group named as an argument to the command.

Syntax	`showGroup <`*`groupname`*` or `*`ID`*`>`
Examples	`> showGroup Finance`
Result	Lists all Identities who are members of the Finance group.

Workflow

The **workflow** command launches the workflow specified as a command parameter. Input variables must be entered in a variable file to get passed to the workflow. A variable file is specified as an XML Map. The file name is then also passed as a parameter to the command. When the workflow is successfully launched, the XML for the workflowCase is printed to the console (stdout).

Syntax	`workflow <`*`workflowname`*` or `*`ID`*`> [`*`varfile`*`]`
Examples	`> workflow “LCM Provisioning” c:\data\provFile.xml`
Result	Runs the LCM Provisioning workflow, passing its input variables through the file c:\data\provFile.xml

The varfile should contain an attributes map like the example shown below. This example map passes an identity name and a provisioning plan object to the workflow.

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE Attributes PUBLIC "sailpoint.dtd" "sailpoint.dtd">
<Attributes>
  <Map>

    <entry key="identityName" value="Adam.Kennedy"/>
    <entry key="plan">
      <value>
        <ProvisioningPlan>
          <AccountRequest application="IIQ" nativeIdentity="Adam.Kennedy" op="Modify">
            <AttributeRequest name="assignedRoles" op="Add" value="PRISM User">
              <Attributes>
                <Map>
                  <entry key="comments" value="req A"/>
                </Map>
              </Attributes>
            </AttributeRequest>
          </AccountRequest>
          <AccountRequest application="IIQ" nativeIdentity="ABC_12345" op="Modify">
            <AttributeRequest name="assignedRoles" op="Add" value="Test Role B2">
              <Attributes>
                <Map>
                  <entry key="comments" value="req B"/>
                </Map>
              </Attributes>
            </AttributeRequest>
          </AccountRequest>
        </ProvisioningPlan>
      </value>
    </entry>
   </Map>
</Attributes>
```

Validate

The **validate** command can validate a workflow or a rule. Input variables must be entered in the variable file to be passed to a workflow or rule. The variable file is specified as an XML map and the file name is passed as a parameter to the command. Validation errors are printed to the console (stdout).

Syntax	`validate <`*`rule`*` or `*`workflow`*` name or `*`ID`*`> [`*`varfile`*`]`
Examples	`> validate “LCM Provisioning” c:\data\provFile.xml`
Result	Validates the LCM Provisioning workflow, passes the input variables through `c:\data\provFile.xml`, and displays any validation errors

See “Workflow” on page 126 for an example of the varfile format.

Wftest

The **wftest** command is used to one or more workflows. The *WorkflowTestSuite* can be the name of a WorkflowTestSuite object or a file containing one.

Syntax	`wftest `*`WorkFlowTestSuite`*` name | `*`filename`*
Examples	> wftest `c:\test\workflowTest.xml` name | `c:\test\workflowTestOut.xml`

Result	Tests the workflows and sends the outcome to the `workflowTestOut` file.

SQL

The **sql** command executes a SQL statement. It can execute SQL specified inline with the command or it can read the SQL from a file. Only one SQL statement can be executed at a time. The output can be printed to the console (stdout) or redirected to a file. Select, update, and delete SQL statements can be executed. Update and delete actions cannot be undone.

Syntax	`sql` *`sqlStatement`* `| -f` *`inputFileName`*

Examples	`> sql “select * from sptr_identity” > c:\data\Identities.dat`
	`> sql -f c:\sql\SelectIdentities.sql`

Result	The first example executes the specified select statement and writes the results to `c:\data\Identities.dat`. The second example reads the SQL from `c:\sql\SelectIdentities.sql`, prints the SQL to the console (stdout), and displays the query results to the console (stdout).

Provision

The **provision** command processes the specified provisioning plan for the specified identity but does not save the information. This is used to test a connector or to test a provisioning plan. Errors are reported to the console. If the provisioning action would succeed, nothing is reported to the console.

Syntax	`Provision <`*`identityname`*` or `*`ID`*`>` *`provisioningPlanFfilename`*
Examples	`> provision Adam.Kennedy c:\data\provFile.xml`
Result	Tests the provisioning plan contained in `c:\data\provFile.xml` against Identity Adam.Kennedy and reports any exceptions to the console

The provisioning plan file should contain a provisioning plan in XML format. For example:

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE ProvisioningPlan PUBLIC "sailpoint.dtd" "sailpoint.dtd">
<ProvisioningPlan>
   <AccountRequest application="IIQ" nativeIdentity="ABC_12345" op="Modify">
     <AttributeRequest name="assignedRoles" op="Add" value="Test Role B1">
       <Attributes>
          <Map>
            <entry key="comments" value="req A"/>
         </Map>
       </Attributes>
     </AttributeRequest>
   </AccountRequest>
   <AccountRequest application="IIQ" nativeIdentity="ABC_12345" op="Modify">
     <AttributeRequest name="assignedRoles" op="Add" value="Test Role B2">
       <Attributes>
         <Map>
           <entry key="comments" value="req B"/>
         </Map>
       </Attributes>
     </AttributeRequest>
   </AccountRequest>
</ProvisioningPlan>
```

Lock

The **lock** command obtains a persistence lock on an object. The object's class and ID or name must be specified. By default, the lock is issued to the username Console, but a different username can be specified in the command's lockName parameter. The lock automatically expires after 5 minutes.

Syntax	`lock` *`classname <objectID`* `or` *`name`*`>` `[`*`lockName`*`]`
Examples	`> lock identity John.Smith`
Result	Obtains a persistence lock for Console on the identity record for John.Smith

Note: Identity objects are the only objects that can be locked. Attempts to specify a different object type in this command results in a syntax error exception.

Unlock

The **unlock** command releases the lock on an object. The object's class and ID or name must be specified. If the object is not locked, the message “Object is not locked” is displayed. If it is locked, the lock is released and the message “Lock has been broken” is displayed.

Syntax	`unlock` *`classname <objectID`* `or` *`name`*`>`
Examples	`> unlock identity John.Smith`
Result	Breaks the lock on the identity record for John.Smith

ShowLock

The **showLock** command lists the lock owner, locked date/time, and lock expiration date/time for a locked object. The object's class and ID or name must be specified to view its lock information. The message Object is not locked is displayed if the object is not currently locked. If the lock has expired, the lock information is shown but is prefaced with the message Lock has expired.

Syntax	`showLock` *`classname <objectID`* `or` *`name`*`>`
Examples	`> showLock identity John.Smith`
Result	Displays lock information (owner, date/time, expiration date/time) on the identity record for John.Smith or displays Object is not locked

Oconfig

The **oconfig** command list all extended attributes defined for each class that supports extended attributes. The list indicates the extended attribute numbers and corresponding attribute names on each class. Identity extended attributes which link to other Identities are stored separately in extended identity attribute fields, so those are listed in a separate Extended Identity Attributes sub-list under the Identity ObjectConfig. The objectConfig detail displays No attributes defined if no extended attributes are defined for a given class. An Object not found message is displayed if no objectConfig exists for the class.

Syntax	`oconfig`
Examples	`> oconfig`

Result	Displays each objectConfig and its extended attributes, numbered according to the extended attribute number that corresponds to each
	`ObjectConfig: Identity`
	`  1 region`
	`  2 Department`
	`  3 location`
	`  4 empId`
	`  5 jobtitle`
	`  Extended Identity Attributes:`
	`    1 regionOwner`
	`    2 locationOwner`
	`ObjectConfig: Link`
	`  1 inactive`
	`  2 service`
	`  3 privileged`
	`  4 lastLogin`
	`ObjectConfig: Application`
	`  1 DeployDate`
	`ObjectConfig: Bundle`
	`  No attributes defined`
	`ObjectConfig: ManagedAttribute`
	`  1 authorization`
	`  2 email`
	`  3 rank`
	`ObjectConfig: CertificationItem`
	`  Object not found`

TextSearch

The textsearch command enables command-line execution of full text searches as they are done through the LCM full text search. The class name must be either ManagedAttribute or Bundle, since those are the only indexed classes. This command searches for the specified string in the fullTextIndex created for the specified class and returns a map representation of the objects in which the string is found. If a filter attribute and value are specified, the search is further constrained to entries that correspond to that attribute name-value pair. The filter is always treated as an equals operation. The filterName must be an attribute that is indexed in the FullTextIndex object for the specified class.

Syntax	`textsearch` *`classname string`* `[`*`filterName filterValue`*`]`

Examples	`> textsearch Bundle manager type business`
Result	Returns a map of data values for each Bundle (role) of type=business that contains the string manager in any analyzed field. Analyzed fields in the Bundle FullTextIndex marked as analyzed=true.

Seldom Used Commands

These commands are used by developers for testing and are rarely used in a production environment.

Properties

The **properties** command displays system properties.

Syntax	`properties`
Examples	`> properties`
Result	Displays Java properties of the server on which IdentityIQ is installed

Time

The **time** command reports the duration of another command.

Syntax	`time` *`command`*
Examples	`> time run “refresh risk scores”`
Result	Initiates the **run** command and then indicates how much time it took to run. Most useful for long-running commands

Xtimes

The **xtimes** command repeats a single command as many times as specified in the first argument. This command is used for performance testing purposes. Running a command numerous time provides a more accurate indication of how long a process takes than running it once.

Syntax	`xtimes` *`x command`*
Examples	`> xtimes 3 run “refresh risk scores”`
Result	Runs the refresh risk scores task three times in a row

Note: This command can be combined with the time command to report timing statistics on the performance test. By specifying this command first (for example, xtimes 20 time run *taskname*), the time taken for each command run is reported. By specifying the time command first (for example, time xtimes 20 run *taskname***), the total time for all of the sequential runs is reported.**

About

The **about** command displays IdentityIQ's application configuration information.

Syntax	`about`
Examples	`> about`
Result	Lists application configuration specifics for the IdentityIQ instance (version, database, host, memory, etc.)

Threads

The **threads** command displays all active threads in the instance.

Syntax	`threads`
Examples	`> threads`
Result	Lists all active threads

LogConfig

The **logConfig** command reloads the log4j configuration into the instance.

Syntax	`logConfig`
Examples	`> logConfig`
Result	Reloads the log4j configuration from the `log4j2.properties` file

Summary

The **summary** command lists all classes and the count of objects of each class. Changes in these counts for some objects (for example, auditConfig) can indicate potential problems or areas of concern.

Syntax	`summary`
Examples	`> summary`
Result	Lists class name and count of objects for each class in the system

Rollback

The **rollback** command can undo a change to a role by restoring it from its BundleArchive object. BundleArchive objects are created when role archiving is enabled for IdentityIQ. Role archiving tracks changes made to a role by storing the pre-modification state in a BundleArchive object when the Bundle object is updated. This command only applies to the BundleArchive class.

Syntax	`rollback` *`classname <objectname`* `or` *`id>`*
Examples	`> rollback BundleArchive “Contractor-IT”`
Result	Restores the Contractor-IT role to the pre-modification state stored in its BundleArchive object

Rename

The **rename** command changes the name of an object from its existing name to the value specified by the *newname* parameter.

Syntax	`rename` *`classname <objectname`* `or` *`ID> newname`*
Examples	`> rename application ADAM ADAM-Production`
Result	Changes the name of the ADAM application to ADAM-Production

Note: The object can be found using its old Name or its ID value, but in either case, the *newname* value is used to update the Name attribute for the object.

ExportJasper

The **exportJasper** command creates a JasperReport XML file from a JasperTemplate object in IdentityIQ. Jasper Report is a third party user interface for report writing. JasperReport XML is not compatible with IdentityIQ's XML so the JasperReport XML is wrapped in a JasperTemplate object when saved in IdentityIQ. The JasperTemplate must be exported to create a file that can be used directly with the Jasper user interface before it can be reformatted.

Syntax	`exportJasper` *`filename <JasperTemplateName`* `or` *`ID>`*
Examples	`> exportJasper c:\data\AggResRpt.xml AggregationResults`
Result	Exports the Jasper XML from the AggregationResults JasperTemplate object into the file `c:\data\AggResRpt.xml`

Note: The import command can be used to re-import a JasperReport object into the database. The import wraps the XML in a JasperTemplate.

Identities

The **identities** command lists the Name, Manager, Roles, and Links for each identity in the system. By default, this information prints to the console (stdout) and can be difficult to read due to screen wrapping. If the output is redirected to a file, it is printed in the file in an easy-to-read style.

Syntax	`identities`

Examples	`> identities`
	`> identities > identities.txt`

Result	The first example writes the Name, Manager, Roles, and Links for each identity in the system to the console (stdout). The second example redirects that information to the file `identities.txt`.

Snapshot

The **snapshot** command takes a snapshot of the named identity as it exists at that moment and archives it in the database as an IdentitySnapshot object. This object provides a historical record of the state of Identity objects at various points in time. Automatic snapshotting can be enabled and configured to create IdentitySnapshot objects at specified intervals or based on system activities (weekly, on aggregation change, etc.). The configuration of this feature can negatively impact system performance.

Syntax	`snapshot <identityname or ID>`
Examples	`> snapshot Alan.Bradley`
Result	Creates an IdentitySnapshot object for the identity Alan.Bradley, capturing his Identity Attributes, Roles (Bundles), Entitlements Outside Roles, Links, and Scorecard information at that moment in time

Score

The **score** command refreshes the identity score for the named identity and updates that score in the database. Score updates are more commonly executed through the IdentityIQ user interface.

Syntax	`score <identityname or ID>`
Examples	`> score Alan.Bradley`
Result	Recalculates the risk scores for Alan.Bradley and updates his Scorecard with the new risk scores

Tasks

The **tasks** command lists the Name, State, Next Execution, and Cron Strings for all currently scheduled tasks in the system.

Syntax	`tasks`
Examples	`> tasks`
Result	All currently scheduled tasks are written to the console (stdout)

TerminateOrphans

The **terminateOrphans** command sets the completion status of any open taskResult objects to Terminated. While tasks are running, their taskResults should be in a pending state, but occasionally task results can become orphaned and remain in this non-completed state when the task has finished (or has otherwise been terminated). This command can be used to clean up those orphaned taskResults but it must only be executed when there are no tasks running on the application server or the taskResults for actively running tasks are terminated along with any orphaned results.

This command requires no arguments for execution but an artificial argument please has been added to prevent accidentally running this command.

Syntax	`terminateOrphans please`
Examples	`> terminateOrphans please`
Result	Sets all open taskResults for the application server to the Terminated status

Certify

The **certify** command creates a manager or application certification. The certification is generated using the installation's default settings/parameters. This command is primarily used for testing purposes.

Syntax	`certify [`*`managerName`*` | `*`application`*`]`
Examples	`> certify Catherine.Simmons`
Result	Generates a manager certification for manager Catherine Simmons

Note: The command syntax help indicates that this command can generate an application owner certification when an application is specified as a command argument, but this feature has not been updated as the certification components of the product have changed over time. As a result, the application argument for this command is not currently usable.

CancelCertify

Note: This command is not recommended. Use the delete command to remove certification objects.

The **cancelCertify** command can be used to delete a certification object from the system.

Syntax	`cancelCertify <`*`certificationName`*` or `*`ID`*`>`
Examples	`> cancelCertify “Manager Access Review for William Moore”`
Result	Delete the named certification (the command fails if more than one certification object with the same name exists)

ArchiveCertification

The **archiveCertification** command archives the specified certification (creates a certificationArchive object) and deletes it as an active certification.

Syntax	`archiveCertification <certificationName or ID>`
Examples	`> archiveCertification “Manager Access Review for William Moore”`
Result	Creates a certificationArchive object and delete the certification from the system

DecompressCertification

The **decompressCertification** command retrieves the named certificationArchive object and prints it to the console (stdout) in the Certification object's XML format.

Syntax	`decompressCertification <certificationArchiveName or ID>`
Examples	`> decompressCertification “Manager Access Review for William Moore”`
Result	Prints the named certification archive to the console (stdout) in certification XML format

WorkItem

The **workItem** command displays certain details (Owner, Create Date, Expiration Date) for the specified workItem.

Note: This command requires the workItem ID or name value as an input parameter. The workItem ID value (a long hexadecimal number) is obtained using the IdentityIQ console's list workItem command. The workItem Name is not the descriptive name for the workitem, it is a numeric value assigned when the workItem is created. The value is found in the XML representation of each workItem through the Debug pages.

Syntax	`workItem <workItemID or Name>`
Examples	`> workItem 40288f0132b155ad0132b58a4e3f018e`
Result	Displays the Owner, Created Date, and Expiration Date for the specified workItem

Approve

The **approve** command sets the specified workItem to a Finished state (indicating it was approved), adds any specified completion comments to the workItem, and submits the workItem to the workflower to move it to the next appropriate stage.

Note: This command requires the workItem ID or name value as an input parameter. You can obtain the workItem ID value (a long hexadecimal number) using the IdentityIQ console list workItem command. The workItem Name is not the descriptive name for the workitem, it is a numeric value assigned when the workItem is created. The value is found in the XML representation of each workItem through the Debug pages.

Syntax	`approve <workItemID or Name> [comments]`

Examples	`> approve 40288f0132b155ad0132b58a4e3f018e “Access approved”`
Result	Marks the specified workItem as approved, adds the comment “Access approved” to the workItem's completion comments, and submits the workItem for evaluation of the next appropriate step (another approval, provisioning, etc.)

Reject

The **reject** command sets the specified workItem to a Rejected state, adds any specified completion comments to the workItem, and submits the workItem to the workflower to move it to the next appropriate stage.

Note: This command requires the workItem ID or name value as an input parameter. The workItem ID value (a long hexadecimal number) is obtained using the IdentityIQ console's list workItem command. The workItem Name is not the descriptive name for the workitem, it is a numeric value assigned when the workItem is created. The value is found in the XML representation of each workItem through the Debug pages.

Syntax	`reject <`*`workItemID`*` or `*`name`*`> [`*`comments`*`]`
Examples	`> reject 40288f0132b155ad0132b58a4e3f018e “Access conflicts with AP data entry entitlement”`
Result	Marks the specified workItem as rejected, adds the comment “Access conflicts with AP data entry entitlement” to the workItem's completion comments, and submits the workItem for evaluation of the next appropriate step (another approval, etc.)

Warp

The **warp** command parses an XML file to create an object and then displays the object's XML representation in the console (stdout). If it is not in valid form or its tags do not match the IdentityIQ DTD, a runtimeException is printed to the console describing the error.

Syntax	`warp `*`filename`*
Examples	`> warp c:\data\newWorkflow.xml`
Result	Parses the XML in the file `c:\data\newWorkflow.xml` and displays the XML representation of the object in the console, or reports any errors to the console

Notify

The **notify** command sends an email message to the specified identity using the email template specified. This command does not accept any other parameters that can be passed to the template, so it can only be used for templates whose messages do not rely on variable substitutions to build the content. This command is most often used for testing purposes.

Note: The toAddress argument can contain an identity name or ID or an email address. If it contains an identity name or ID, the email address is retrieved from the identity record.

Syntax	`notify <`*`emailTemplateName`*` or `*`ID`*`> `*`toAddress`*

Examples	`> notify Certification Alan.Bradley`
Result	Sends an email to Alan.Bradley's email address using the Certification email template

Authenticate

The **authenticate** command authenticates a username and password against the pass-through authentication source or the internal IdentityIQ records. No results are returned if the values are authenticated. If the password is incorrect or the user name cannot be found, an error message is displayed in the console (stdout).

Syntax	`authenticate` *`username password`*
Examples	`> authenticate Alan.Bradley s53n659#@5a!`
Result	Authenticates username Alan.Bradley and the provided password against the authentication source (pass-through or internal)

SimulateHistory

The **simulateHistory** command is used to generate a fake, randomly-generated group index or identity score history for one or more groups or identities. Used for generating test data in a development environment.

Syntax	`simulateHistory Identity|Group <`*`groupName`*` or `*`ID`*`>|<`*`identityName`*` or `*`ID`*`>|all`

Examples	`> simulateHistory Identity all`
	
	`> simulateHistory Group Finance`

Result	First example generates fake risk scorecards for all identities in the system Second example generates fake groupIndex information for the Finance group

Search

The **search** command looks up an object based on specified criteria, similar to a simplified SQL/HQL interface. A single class name is specified with a list of the attributes to display from that class. Following the `where` keyword, search filters can be specified in name value sets. All filter values are used in a like comparison. The record is returned if the record's field value contains the specified value string.

Syntax	`search `*`className`*` [`*`attributeName…`*`] where [`*`filter…`*`]`
	
	`filter: `*`attributeName value`*

Examples	`> search identity name manager.name region where name kat`
Result	Returns the name, manager's name, and region for all identities whose name contains the string kat. For example, records for Katherine.Jones, John.Kato, and Tammy.Erkatz are returned by this search

CertificationPhase

The **certificationPhase** command transitions the specified certification to the specified phase. This command fails if the certification is on or past the requested phase.

Note: The certification is advanced to the next enabled phase after the requested phase if the specified phase is not enabled for the certification. For example, if a certification has neither a Challenge nor a Remediation phase enabled but the command requests that it be advanced to the Challenge phase, the certification is advanced to the End phase.

Note: The certification is sequentially advanced through all enabled phases until it reaches or passes the requested phase. Any business logic that should occur during each phase transition (period enter rules, period end rules, etc.) is executed during the phase advancement.

Syntax	`certificationPhase <`*`certificationName`*` or `*`ID`*`> [Challenge | Remediation | End]`
Examples	`> certificationPhase “Catherine Simmons Access Review” Challenge`
Result	Advances the “Catherine Simmons Access Review” certification from its current phase (Active) to the Challenge phase. If this review is not configured for a Challenge phase, it is transitioned to the Remediation or End phase (depending on configuration).

Impact

The **impact** command reads an XML file containing a Bundle (role) object and performs role impact analysis for the role. The command parses the XML to its object form. Impact analysis is not performed if that object is not a Bundle.

Syntax	`impact `*`filename`*
Examples	`> impact c:\data\ContractorRole.xml`
Result	Performs role impact analysis for the Bundle object represented by the XML in `c:\data\ContractorRole.xml`

Event

The **event** command schedules a workflow to run, passing in an Identity name as an argument. By default, the workflow is scheduled 1 second after the command is issued, but a delay can be specified in seconds as a command argument.

Syntax	`event <`*`identityName`*` or `*`ID`*`> <`*`workflowName`*` or `*`ID`*`> [`*`seconds`*`]`
Examples	`> event Catherine.Simmons “Identity Refresh” 60`
Result	Schedules an Identity Refresh workflow to run for Catherine.Simmons 60 seconds after the command is issued

ConnectorDebug

The **connectorDebug** command is used to test a connector or troubleshoot application aggregation issues. Its method parameters determine what is tested and how.

Syntax	`connectorDebug <applicationName or ID> <method> [methodArgs…]`

The specific syntax for each of the “methods” is shown below.

Method	test
Purpose	Test whether a connection can be established with the application through its connector
Syntax	`connectorDebug <applicationName or ID> test`
Example	`> connectorDebug ADAM test`
Result	Returns “Test Succeeded” on success, reports an error in the console on failure.

Method	iterate
Purpose	Iterate through the application's account or group records
Syntax	`connectorDebug <applicationName or ID> iterate [account|group (default = account)] [-q (for "quiet mode")]`

Example	`> connectorDebug ADAM iterate -q`
	`> connectorDebug ADAM iterate account`

Result	First example iterates all account records natively in the ADAM application and returns only the count of iterated objects and how many milliseconds it took to run. Second example iterates account records natively in the ADAM application and returns a ResourceObject representation of each account to the console.

Method	get
Purpose	Test whether a connection can be established with the application through its connector
Syntax	`connectorDebug <applicationName or ID> get account|group nativeIdentity`
Example	`> connectorDebug ADAM get account “CN=Willie.Gomez,DC=sailpoint,DC=com”`
Result	Returns the XML representation of the ResourceObject for that nativeIdentity on the application

Method	auth

Purpose	Test pass-through authentication against the specified application (The featuresString in its application definition must contain AUTHENTICATION.)
Syntax	`connectorDebug <applicationName or ID> auth username password`
Example	`> connectorDebug ADAM auth administrator Pa$$w0rd`
Result	Returns "Authentication Successful" when user is authenticated or displays the exception message to the console if authentication fails

Encrypt

The **encrypt** command is used to encrypt a string. This command is generally only useful for test purposes. It can generate an encrypted password which can be passed in other console commands, for example, the **authenticate** command.

Syntax	`encrypt string`
Examples	`> encrypt MyPa$$w0rd`
Result	Returns the encrypted equivalent for the specified string

HQL

The **hql** command executes a search based on a Hibernate Query Language statement. The command syntax matches the sql command's syntax, but this command can select but not update data.

Syntax	`hql hqlStatement | -f inputFileName`

Examples	`> hql "select name, manager.name from Identity" > c:\data\Identities.dat`
	
	`> hql -f c:\hql\SelectIdentities.hql`

Result	The first example executes the specified HQL select statement and writes the results to the file `c:\data\Identities.dat`. The second example reads the HQL from the file `c:\hql\SelectIdentities.hql`, prints the HQL to the console (stdout), and displays the query results to the console (stdout).

Date

The **date** command shows the current date and time for the application server or the date and time value for a specified utime (universal time) value.

Syntax	`date [utime]`

Examples	`> date`
	
	`> date 1338820492484`

Result	The first example displays the command syntax and the current date/time and current UTIME value. The second example returns the date/time value for the specified UTIME value.

Shell

The **shell** command escapes out to the command line and runs the command specified. (This command does not work properly in a Windows environment but does work in UNIX.

Syntax	`shell` *`commandLine`*
Examples	`> shell ls`
Result	Lists the contents of the UNIX file system directory from which the console was run

Meter

The **meter** command toggles metering on or off. While metering is on, the console reports some timing statistics for each command executed. Meter information is displayed after the results of each command as it is executed.

Syntax	`meter`
Examples	`> meter`
Result	Toggles metering on and off. When turned on, all subsequently issued commands report timing statistics. Meter information displayed includes: number of calls, total number of milliseconds, maximum time for one call, minimum time for one call, and average time per call.

Compress

The **compress** command is designed to compress the contents of a file to a string that can be included within an XML element. It compresses the file and then encodes it to Base64 and writes that text to the specified output file. This resultant file can then be used in an XML element stored in the database. This has limited usefulness within IdentityIQ since no part of the application is designed to read these compressed strings, but custom rules can be used to process them as needed or they can simply be stored in the database to be retrieved and uncompressed for use by an external application at a later time.

Syntax	`compress` *`inputFilename outputFilename`*
Examples	`> compress file1.txt file2.txt`
Result	Compresses the contents of `file1.txt`, encodes that into Base64, and writes the resultant text string to `file2.txt`

Uncompress

The **uncompress** command functions in exactly the opposite way of the **compress** command, taking a compressed, Base64-encoded file and returning its uncompressed format.

Syntax	`uncompress` *`inputFilename outputFilename`*
Examples	`> compress file2.txt file3.txt`
Result	Reverses the compressing process to return the original, uncompressed version of the text, writing that to the file `file3.txt`

ClearEmailQueue

The **clearEmailQueue** command deletes all queued but unsent email messages from the IdentityIQ email queue. This includes any new messages that have not yet been sent and messages that have encountered problems that prevented successful delivery.

Syntax	`clearEmailQueue`
Examples	`> clearEmailQueue`
Result	Deletes all unsent emails from the email queue

ClearCache

The `clearCache` command removes objects from the Hibernate object cache. This can be used when debugging Hibernate issues.

Syntax	`clearCache`
Examples	`> clearCache`
Result	Clears the Hibernate object cache

Service

The **service** command provides information about the background services running in the console. The services include:

- Cache - periodically refreshes cached objects
- SMListener - listens for change events from PE2 change interceptors
- ResourceEvent - looks for change events added to a queue and processes them
- Heartbeat - maintains a Server object for each IdentityIQ instance and periodically updates it so you can tell if an instance is still running
- Task - the Quartz task scheduler
- Request - the IdentityIQ request processor - stopping the Request service also stops partitioned tasks

Syntax	`service list	start	stop	run`
Examples	`> service list`
Result	Lists background services running in the console

Chapter 9: Classifications

Classifications let you flag and categorize roles and entitlements, to help ensure the security and integrity of your access governance practices. Classifications can alert you when requesting, granting, or approving a user’s access will give that user access to sensitive, protected, or otherwise significant data.

In IdentityIQ, classifications are typically used to flag access to sensitive data, such as financial, personal, or health-related information, but you can use classifications to identify any kind of access your business needs to pay special attention to.

Classifications can be used in certifications and policies, to help you monitor and control the access your users have to sensitive data. You can configure access requests, approvals, and access reviews to show a classification icon with any role or entitlement that grants access to sensitive data, so that the users responsible for making access decisions can quickly and easily see which entitlements allow potentially risky access.

This chapter includes the following sections:

Where Classification Data Comes From

IdentityIQ’s classification functions are designed to integrate with SailPoint’s File Access Manager module, to provide robust and seamless governance of sensitive data.

You can also implement classifications using data from sources other than File Access Manager, to tailor your classifications solution to your particular business needs.

File Access Manager Classifications

In File Access Manager, classification categories are assigned to Business Resources (folders). Classifications typically flag sensitive data, but can flag anything you configure File Access Manager to monitor. In many cases, access to classified data is granted through account groups - most typically, Active Directory Groups - so that users' membership in those groups indirectly grants access to that data. For example, a company’s Human Resources group might have access to employees’ personal data, or a hospital’s group of doctors might have access to medical records. When you include File Access Manager's classification data in your IdentityIQ installation, you can see the implications of users' existing (and requested) group memberships, to better inform your access governance decisions in IdentityIQ

For more information about how data is classified in File Access Manager, refer to the *IdentityIQ File Access Manager Administrator Guide*.

Classifications from Other Sources

You can bring classification data into IdentityIQ from sources other than File Access Manager, or define your classification data independently in IdentityIQ, by importing the classification data as an XML object, using the iiq console or the **gear menu > Global Settings > Import From File** feature.

Here is an example of what a classification object might look like. This example includes a name, display name, source of origin for the data, and localized descriptions for the classification.

```
<?xml version='1.0' encoding='UTF-8'?>

<!DOCTYPE Classification PUBLIC "sailpoint.dtd" "sailpoint.dtd">

<Classification id="" name="PHI" displayName="Protected Health Information"
origin="MyIndependentDataSource">

  <Attributes>

    <Map>

      <entry key="sysDescriptions">

        <value>

          <Map>

           <entry key="en_US" value="Allows access to Protected Health Information"/>

           <entry key="fr_FR" value="Permet l’accès aux informations de santé
protégées"/>

          </Map>

        </value>

      </entry>

    </Map>

  </Attributes>

</Classification>
```

Working With Classification Data in IdentityIQ

Classifications in IdentityIQ are managed as attributes on entitlements - if you are integrating with File Access Manager, these entitlements will most typically be *group* entitlements. For example, a Human Resources group is aggregated into IdentityIQ as a group entitlement; if this group is categorized in File Access Manager (or some other source) as having access to sensitive information, an attribute that flags the Human Resources group entitlement as having this access is added to the group entitlement. Once you have defined classifications in IdentityIQ, you can apply classification attributes to *any* entitlement, not just group entitlements.

Entitlements are managed in IdentityIQ, using IdentityIQ’s range of compliance and lifecycle management features, such as access requests, certifications and access reviews, policies, and reporting.

You can view and manage classifications in these areas of IdentityIQ:

Lifecycle Manager: Access Requests and Approvals

A global setting in Lifecycle Manager determines whether classification data is shown with the access items (such as roles or entitlements) that you can request for users in the Manage Access feature. This global setting is provided so that you can choose whether or not to alert requesters to the fact that certain roles or entitlements may allow access to sensitive or protected data.

To enable the display of classifications in Access Requests:

1. Click the **gear menu > Lifecycle Manager.**

2. On the **Configure** tab, scroll to the **Manage Classifications Options** section.
3. Check the **Display classifications in Access Request** box.
4. **Save** your change.

If this setting is enabled in Lifecycle Manager, roles and entitlements are flagged with relevant classification information in the **Access Requests** pages. You can click the **Details** button for flagged roles and entitlements, to see more information about the classifications.

Classification data also appears in the **Approvals** page for access requests. Classification flags always appear in the **Approvals** page, regardless of the setting in the Lifecycle Manager's **Manage Classifications Options** section, since reviewers will always need to know when granting access will allow access to sensitive or protected data.

Adding Classifications to Roles and Entitlements

For File Access Manager integrations, classifications can be added to entitlements by running a task. This process is described in more detail in "File Access Manager Classification Processes" on page 150.

For classifications that come from a source other than File Access Manager, classifications can be added manually to roles and entitlements.

To add classifications to a role:

1. Click **Setup > Roles**.
2. To add classifications to existing roles, find the role you want to edit in the **Role Viewer,** then click **Edit Role**; for new roles, click **New Role > Role**.
3. In the **Role Editor**, select the classifications you want to add from the drop-down list.
4. **Save** your changes.

You can also include classifications as criteria in "Match List" **Assignment Rules** for the role. Assignment Rules are used to automatically assign roles to identities during a correlation process.

In the **Role Search** tab you can include classifications as search criteria.

To add classifications to an entitlement:

1. Click **Applications > Entitlement Catalog**.
2. To add classifications to existing entitlements, use the **Filter** field or **Advanced Search** to find the entitlement you want to edit; for new entitlements, click **Add New Entitlement.**
3. On the **Classifications** tab, select the classifications you want to assign from the drop-down list, then click **Add** to assign the classification.
4. **Save** your changes.

In the **Advanced Search** feature of the Entitlement Catalog, you can include classifications as search criteria.

Classifications in Certifications and Access Reviews

When scheduling a certification campaign, you can opt to show classification data the campaign's access reviews. Classifications can be shown in Manager, Application Owner, Advanced, Role Membership, and Targeted certifications. You can also use classifications as a criterion for what to certify, in Targeted certifications.

You can set a global default to show classifications for all your certification campaigns, and modify the default setting in any individual certifications you schedule.

To set the global default for showing classifications in your certification campaigns:

1. Click the **gear menu > Compliance Manager**.
2. In the **Behavior** section, use the **Show Classifications** checkbox to enable or disable showing classifications by default.

CAUTION—When Show Classifications is enabled here, any Separation of Duties (SOD) Policy Violations can include information about classifications in the Revoke > Correct Violations dialog. If you disable Show Classifications here, classification details will not appear in that dialog.

3. **Save** your changes.

To modify the default behavior in an individual certification, check or uncheck the **Show Classifications** checkbox in the **Schedule Certification** page.

- In Manager, Entitlement Owner, Advanced, and Role Member certifications, this option is on the **Behavior** tab.
- In Targeted certifications, this option is in the **Additional Settings** section, under **Advanced Options**.

To include classifications as a criteria for what to certify in a Targeted certification:

1. In the **What to Certify** section, click **Filter Roles** or **Filter Entitlements**.
2. Select **Classifications** as an attribute.
3. Choose an operator (**Equals** or **Not Equals**) and select your classifications from the drop-down.

Classifications in Policies and Policy Violations

In **Advanced** policies, you can use classifications as criteria for your policy rules.

To add classifications to an Advanced policy rule:

1. Click **Setup > Policies**.
2. To add classifications to an existing policy, use the **Filter** field or **Advanced Search** to find the policy you want to edit; for new policies, click **New Policy** > **Advanced Policy.**
3. Click **Create New Rule**, or double-click an existing rule you want to edit. Classifications can be used as rule criteria in Match List, Rule, Script and Filter rules.

   Note: Rules and scripts are written in BeanShell, and Filters are an XML specification.

4. For Match List rules:
   a. Under **Selection Method**, choose **Match List**.
   b. Click **Add Role Attribute** or **Add Entitlement Attribute**.
   c. In the **Name** field choose **Classification**.
   d. Choose an operator: **Equals**, **Not Equals**, or **Is Null**.
   e. In **Value**, type the name of the classification to use. (To find the name of a classification, you can use the Debug pages to open the classification object and find the `name` value.)
5. When you have added all the classification criteria you want to use, you can run a simulation of the rule, or click **Done** to save your changes and exit.

Classifications in Advanced Analytics

In the Advanced Analytics page, you can search for roles and entitlements using classifications as search criteria.

1. Click **Intelligence > Advanced Analytics**.

2. Choose **Role** or **Entitlement** as the **Search Type**.
3. Choose a classification to search on, from the drop-down.
4. If you want to see classification details in your search results, select **Classifications** in the **Fields to Display** panel.
5. Click **Run Search**.

Classifications in the Identity Warehouse

To see a user's classifications in the Identity Warehouse:

1. Click **Identities > Identity Warehouse**.
2. Select an identity.
3. Click the **Entitlements** tab. Any entitlement or role with a classification assigned to it is flagged with the classifications icon.

Classifications in the Edit/View Identity Page

The Manage Identity feature shows classifications for entitlements on identities.

1. In the Quicklinks menu, click **Manage Identity**
2. Choose **Edit Identity** or **View identity**.
3. Click on the identity; the **Access** panel for the identity shows a classification icon for any entitlements with classifications assigned. Click the classification icon for more details.

Integrating with File Access Manager for Classifications

For integration with File Access Manager's classification feature, the initial installation and configuration involves two steps:

1. Import the `init-fam.xml` file into IdentityIQ, using the iiq console or the **gear menu > Global Settings > Import From File** page.
2. Click the **gear menu > Global Settings > File Access Manager Configuration.**

Table 45—File Access Manager Configuration Settings

Field Name	Description
File Access Manager Hostname	The hostname of the File Access Manager web client server. For example, https://webclient.mydomain.com
Basic	Select this option for basic authentication to the File Access Manager web client, using a username and password.
OAuth	Select this option for OAuth authentication to the File Access Manager web client, using a Client ID and Client Secret.
Username	For Basic authentication: the username for logging in to the File Access Manager web client.

Table 45—File Access Manager Configuration Settings (Continued)

Field Name	Description
Password	For Basic authentication: the password for logging in to the File Access Manager web client.
Client ID	For OAuth authentication: the Client ID for logging in to the File Access Manager web client. This value is stored in the web client in **Settings > General > API Authorization**.
Client Secret	For OAuth authentication: the Client Secret for logging in to the File Access Manager web client. This value can be copied from the web client in **Settings > General > API Authorization**.
SCIM Correlation Rule	If the correlation logic in your configured applications does not meet your needs for correlating File Access Manager groups and accounts against IdentityIQ groups, you can use a custom rule to manage correlation. The rule must have a rule type of `Correlation` in order to appear in this drop-down.
SCIM Correlation Applications	Select the applications to correlate File Access Manager groups and accounts against. Typically these will be Active Directory applications.

If you are implementing classifications that come from a source other than File Access Manager, you do not need to take any special steps to configure the feature. You can import your classification objects directly into IdentityIQ and manage classifications as described in the sections above.

File Access Manager Classification Processes

Bringing classification data from File Access Manager into IdentityIQ, and including classifications in your lifecycle and data governance practices, is a multi-step process. An overview of these processes is provided here.

This section assumes you have already completed the configuration in File Access Manager to classify resources and identify which groups have access to those resources. It also assumes that you have applications configured in IdentityIQ for aggregating group and account data.

When you work with classifications that originate in File Access Manager, the assumption is that both the IdentityIQ instance and the File Access Manager instance use the same group data. If this is not the case, you may need to configure rule-based logic to correlate your File Access Manager accounts and groups with your IdentityIQ accounts and groups. You can specify a custom correlation rule for this aggregation in **Global Settings > File Access Manager Configuration**, in the **SCIM Correlation Rule** field.

At a high level, these are the steps for aggregating and managing classifications from File Access Manager.

Application Configuration

1. Configure the IdentityIQ application(s) that aggregate group data. As part of this configuration, you must specify a correlation key in each application’s group schema, to correlate groups in IdentityIQ to groups in File Access Manager.

   For Active Directory applications, the group schema attribute to set as the correlation key is **MsDs-PrincipalName**.
2. In the **File Access Manager Configuration** (under the **gear menu > Global Settings**), add each of the applications that aggregate group data to the **SCIM Correlation Application** field.

Run Tasks to Aggregate and Process Classification Data

IdentityIQ uses tasks to aggregate accounts, groups, and File Access Manager classification data. If you do not already have tasks set up to aggregate accounts and groups, you will need to set these tasks up as part of implementing this feature. You must also create and configure a File Access Manager Classification task. For more detail about setting up tasks see "Tasks" on page 65.

These tasks should be run on a recurring basis, to keep your classification data in IdentityIQ current.

1. Run a task to aggregate **groups**. Typically these will be Active Directory groups.
2. Run a **File Access Manager Classification** task. This is a new task type that was introduced in IdentityIQ version 8.1 as part of the classifications feature. Full details about configuring the options for this task is in "Tasks" on page 65.
3. *Optional*: Run an **Effective Access Indexing** task. You only need to run this task if you are tracking classification data for effective access items. These options are important for managing classifications:

   - **Index classifications**: use this option to add an entitlement’s classifications to the target association that is created when the entitlement target is indexed; in the UI, this means that an entitlement’s classifications will be displayed whenever that entitlement occurs as Effective Access.

   - **Promote classifications**: use this option to promote classifications up the effective access "chain" to the entitlement that grants the effective access. For example, if EntitlementA grants you effective access to EntitlementB, and EntitlementB has a classification assigned to it, then with the Promote Classifications option enabled, the classification assigned to EntitlementB will also be displayed in the UI for EntitlementA.

Chapter 10: Using the Administrator Console

Use the Administrator Console link, under the gear icon, to access the Administrator Console and view the Task, Provisioning, and Environment monitoring tables.

- "Manage Task Results" on page 153
- "Manage Provisioning Transaction Results" on page 154
- "Monitoring Your Environment" on page 155

Access to the Administrator Console is controlled with IdentityIQ rights.

Manage Task Results

Use the Tasks table to view host affinity check run time data. From this page you can also postpone a scheduled task, terminate a running task, or dump a stack trace of a running task. The stack trace is typically used when the task is running long and to diagnostics is desired.

Use the tabs at the top of the table to limit your view by task status: Active, Scheduled, or Completed. Use the **Filter** options or search field to further limit the tasks displayed.

Active Tab

This tab displays all of the tasks that are currently running.

Use the Actions column to terminate a running task or request a stack trace, if a task is running long and you would like to see diagnostics.

Table 46—Manage Tasks - Active Tasks Tab

Column	Description
Name	Name of the task
Type	Task type
Start Date	Name of the task
Owner	The task owner, not necessarily the identity who requested the task be run
Host	Host on which the task is currently running
Current Runtime	How long the task has been actively running
Average Runtime	The time that this task has historically taken to complete

Scheduled Tab

This tab displays all of the tasks that are scheduled to run in the future, including those that are scheduled to run periodically, for example Perform Maintenance.

Use the Actions column to postpone a scheduled task or delete a schedule. No instance of a postponed task will be performed until after the selected date.

Table 47—Manage Tasks - Scheduled Tasks Tab

Column	Description
Name	Name of the task
Type	Task type
Task	Name of the task
Host	Host on which the task is scheduled to run
Next Execution	The next time this task is scheduled to run
Last Execution	The last time this task was executed
Last Result	The result of the last run, for example Success or Failed
Owner	The task owner, not necessarily the identity who requested the task be run

Complete Tab

This tab displays all of the tasks that have completed, regardless of the result.

Table 48—Manage Tasks - Completed Tasks Tab

Column	Description
Name	Name of the task
Type	Task type
Result	The result of the last test run
Start Date	The date and time at which this task began
Date Complete	The date and time at which this task stop running
Owner	The task owner, not necessarily the identity who requested the task be run
Host	Host on which the task was run
Average Runtime	The time that this task has historically taken to complete
Runtime	The actual runtime
Diff from Average	The difference between the actual and average runtimes

Manage Provisioning Transaction Results

Note: This feature can be disabled and might not appear in your instance of IdentityIQ. Contact your system administrator for details.

Use the Provisioning Transactions table to view the status of all provisioning transactions in your implementation of IdentityIQ; connectors, manual work items, and IdentityIQ operations.

Use the tabs at the top of the table to limit your view by transaction status: All, Failure, Success, or Pending. Use the **Filter** options or search field to further limit the transactions displayed. The logging level is controlled by a system setting. If you are not seeing all of your transactions, contact your system administrator.

Use the report/download button to launch a Provisioning Transaction Object report in the background. From the Report Launched window, use **Get Email Notification** to receive an email when the report is complete, or **View Report** to display the Report Results page.

Click the information icon for any transaction to view detailed information. The Transaction Details window provides very detailed information, including the reasons for a Failed or Pending status. After viewing take the appropriate actions to correct the reasons for the failure or delay, you can use the **Override** or **Retry** options to proceed with the provisioning process.

Use **Override** to manually create a work item to take action on failed transactions.

Use **Retry** to manually retry the provisioning transaction. The retry option is only available on transactions that are in the Pending state, and only for transactions that were created with the retry options enabled. This button overrides the reset counter configured in the transaction.

Failed transaction cannot be retried, you must use the override option to create a new work item for those transactions.

The information contained on this page is also available in two reports, the Provisioning Transaction Object Report and the Detailed Provisioning Object Report.

Manage the information displayed on this page from the Miscellaneous tab of the Configure IdentityIQ Settings page of Global Setting, found under the gear icon.

Monitoring Your Environment

The Environment Monitoring page provides insight into each defined Application's health and the status of your Modules and Extensions. This helps diagnose issues with connectivity within the environment.

The Application view provides a view from an Application up perspective. Each Application is listed, along with a summary of all statuses reported by all configured Hosts.

Click **Columns** to select and arrange the information displayed on the pages. Use the search field to locate specific information.

Use the gear icon in the title bar to define global settings for all hosts in IdentityIQ. These settings are used for all hosts that have not explicitly over-ridden the defaults.

Hosts

This tab displays all of the hosts associated with an IdentityIQ instance.

Click on a name in the **Host Name** column to show all ServerStatistics captured for the selected host, grouped by Snapshots. The snapshots can be cycled using the previous/next arrows, or selected by name using the drop-down list.

Use the action buttons in the **Host Action** column to configure or delete hosts. The Host Setting dialog enables you to specify the services running on each host and configure host monitoring.

Deleting a host will remove all associated server statistics, as well as the Server object. The host will no longer appear in the list of hosts after deletion. However, if the underlying server is still running, the host will reappear the nest time its heartbeat service runs. All configuration settings for a re-generated host will use defaults for the its list of services, and for the monitoring service configuration.

Services

This enables a specific host to enable/disable services. The one exception is the Request Service. The Request Service cannot be fully shut down. The Host, if service shutdown, still processes requests that have been specifically targeted to that host, but will not pick up generic requests (un-targeted requests).

Configuration

Note: The Application Monitoring does not adhere to the restore defaults.

The Configuration tab enables host specific monitoring configuration. This enables you to override the global defaults for Polling Interval and Statistics Retention, and to enable and disable given retained statistics.

Click **Use Default Settings** to clear all host specific overrides revert back to the global defaults.

The Configuration tab also allows selecting Applications in which to monitor health. When selected, the Application is contacted each time the monitoring service runs, and the health check status is recorded.

Applications

Note: An application must be monitored by at least one host before it will report statistics.

The Application tab provides a view from an application up perspective. Each application is listed, along with a summary of all statuses reported by all configured hosts. Monitoring can be run from any number of servers, on any subset of applications.

Click an application name to display a panel containing more detailed information about a the application's statuses. This shows each host that has reported a status for the application, as well as the status, and time of ping.

If you have full access rights, click the refresh icon to schedule a request on the host to perform a health check for the application. The refresh icon is disabled until the request is fulfilled.

SailPoint Modules and Extensions

The SailPointModules and Extensions tab provides a list of all installed modules and extensions and a summary of all statuses reported. Click a module or extension name to see a list of reported statuses.

Managing Business Processes

This section contains the following information:

Chapter 11: Business Process Management

A Business Process is a sequence of operations or steps that are launched to perform work. IdentityIQ Business Processes include standard “out-of-the-box” processes and custom “installation-specific” processes. System events trigger both standard and custom IdentityIQ Business Processes. The informal term, workflow, is used in this section to refer to a business process.

The following events can trigger a workflow:

- Role creation or modification
- Account Group creation or modification
- Identity update
- Identity refresh
- Identity correlation
- Deferred role assignment, de-assignment
- Deferred role activation, deactivation
- Any Lifecycle Manager event
- Any Lifecycle Event (marked by changes to an Identity's attributes)

Custom workflows can be defined to do a wide variety of processing tasks. You can use:

- IdentityIQ workflow library methods and rules.
- Custom BeanShell scripts and rules.

Customizing or creating workflows generally involves a combination of XML and Java/BeanShell programming. You can manage some customization activities with the IdentityIQ graphical process editor that is included in the product. To customize or create new workflows, typically you need to be comfortable writing XML and Java.

This section has the following topics:

- "Workflow Basics" on page 161
- "Using the Business Process Editor with Workflows" on page 167
- "Editing Workflow XML" on page 183
- "Advanced Workflow Topics" on page 221

Chapter 12: Workflow Basics

This section contains some key concepts for developing and using workflows.

Terminology

The terms, Business Process and Workflow, are used synonymously In IdentityIQ and throughout this document. The IdentityIQ user interface refers to these sets of connected actions as Business Processes, which is the term that business managers often use.

Important Workflow Objects

The IdentityIQ Object Model uses four key objects in workflows. To work with workflows, you need a basic understanding of these objects.

Table 49—Important Workflow Objects

Object	Usage
Workflow	Defines the workflow structure and steps involved in the workflow processing.
WorkflowCase	Represents a workflow in progress. Contains a workflow element in which the process is outlined and current state data is tracked. Contains identifying information about the workflow target object.
WorkflowContext	Tracks launchtime information the Workflower maintains as it advances through a workflow case. Passed into rules and scripts and to the registered WorkflowHandler. Contains all workflow variables, step arguments, current step or approval, workflow definition, libraries, and WorkflowCase.
TaskResult	Records the completion status of a task, or in this case, the workflow. Contained within the WorkflowCase.

Note: The most important object for writing workflows is the WorkflowContext object, which tracks the launchtime state of the workflow and performs other critical functions. Because WorkflowContext methods are used in workflows, data can be extracted from it as needed within any step of the workflow.

Workflows Operation

Workflows carry out a sequence of defined actions based on a triggering event and can be used for a variety of activities within the system. In its launching state, a workflow is tracked through a workflow case, which manages only one target entity at a time (one identity, one role, one provisioning plan, etc.).

Note: If multiple identities are modified at one time in a way that requires a workflow to launch for all of the identities, a separate workflow case is created to track the processing of the workflow for each single identity.

Provisioning Plans in Workflows

A provisioning plan contains a list of requested changes to an identity. Most workflows that change identities contain a single provisioning plan in a workflow variable. When performing Workflow customization you commonly need to inspect and sometimes need to modify the provisioning plan.

Note: Only one provisioning plan can be referenced in a workflow case at a time.

When you request changes for more than one identity at a time, even if the same change is requested for all the identities:

- A separate provisioning plan is created for each identity.
- A separate workflow case is created to manage the provisioning plan created for each identity.

Triggering Workflows

Events that occur in other parts of IdentityIQ and changes to attributes can trigger Workflows. Common Workflow triggers include the following items:

- **Lifecycle Manager Actions** — Requests to change an identity's roles, entitlements, or accounts can activate workflows.
- **Lifecycle Events** — Creating an identity, deactivating an identity, or moving an identity from one manager to another manager can activate workflows.
- **Non-Lifecycle Events** — Editing a role, editing an account group, and changing a password can active workflows.
- **Identity Attribute Change** — Value changes can activate workflows.
- **Policy Violations** — A policy violation can activate workflows.

The following table lists the four main areas of IdentityIQ where you can associate Workflows to system activities.

Table 50— IdentityIQ Setup for Workflows

Workflow Trigger	IdentityIQ Setup
Lifecycle Manager Requests	Select **Lifecycle Manager** from the gear icon menu and go to the Business Processes tab.
Lifecycle Events	Select **Lifecycle Events** from the **Setup** menu and specify the business process behavior.
Non-LCM-related Events	Linked to triggering events. Select **Global Settings** from the gear icon menu and go to the IdentityIQ Configuration page. Select the Identities, Roles, or Miscellaneous tab and then select a business process.

Table 50— IdentityIQ Setup for Workflows

Workflow Trigger	IdentityIQ Setup
Identity Attribute Change	Configured with a Value Change Workflow. Select **Global Settings** from the gear icon menu and go to the Identity Mapping page. Click an attribute to edit or add a new attribute. On the Edit Identity Attribute page, go to the **Advanced Options -> Value Change Workflow** option to select the business process.
Policy Violation	Select **Policies** from the **Setup** menu, select or create a new policy, and specify the business process behavior

Note: You can also configure an IdentityIQ task to trigger a workflow. This workflow set up is a more complex process. See "Advanced Workflow Topics" on page 221.

IdentityIQ Default Workflows

IdentityIQ is preconfigured with various standard workflows that manage activities. The following workflows are examples of default workflows that are included with the product:

- Provisioning of roles or entitlements
- Account management
- Identity creation
- Password management

The default workflows can be configured and customized to address the specific business requirements of each installation. Additionally, you can write new workflows and apply them to any of the actions in IdentityIQ that support workflows.

Workflow Types

Default workflows have pre-defined workflow types. IdentityIQ uses these assigned types to determine which workflows to present in the Business Process configuration list boxes. Workflows can be specified to activate based on a specific system event.

For example, role create, update, and delete actions can trigger a **RoleModeler** type of workflow. Only workflows of that type are listed in the drop-down list for that configuration option.

Note: You can assign custom types to workflows. However, custom type workflows can only be triggered through the user interface on Lifecycle Events, which can trigger workflows of any type.

The table below lists the workflow type associated with each type of action within IdentityIQ.

Table 51—Workflow Types

Process Type	Description
Policy Violation	Workflow activated to launch policy violation actions.
Batch Provisioning	Workflow activated to launch batch requests.
Scheduled Assignment	Workflow activated to when a role is ready to be assigned.

Table 51—Workflow Types

Process Type	Description
Scheduled Role Activation	Workflow activated when a role is ready to be enabled or disabled.
Managed Attribute	Workflow activated when an entitlement is created or edited.
Identity Correlation	Workflow activated when performing identity correlation tasks.
Identity Event	Workflow activated for identity event. For example, sunrise/sunset dates for deferred entitlement, role assignment, or role removal.
Identity Lifecycle	Workflow activated for Lifecycle events. For example, Lifecycle Event - Joiner or Lifecycle Event - Leaver.
Identity Update	Workflow activated when you update an identity through the **Identity** -> **Identity Warehouse** page. Typically requires few or no approvals.
Identity Refresh	Workflow activated for identities that are refreshed using the Identity Refresh task. This type of process can be used for additional customization during refresh, and to present provisioning policy forms if accounts need to be created as a result of automated role assignment.
LCM Identity	Workflow associated with Lifecycle Manager Identity related tasks, for example, LCM Create and Update.
LCM Provisioning	Workflow activated for Lifecycle Manager provisioning tasks.
LCM Registration	Workflow activated for registration tasks.
Policy Violation	Workflow activated to initiate policy violation actions.
Role Modeler	Workflow associated with Role functions. For example, Owner Approval and Role Activation.
Subprocess	Designation of a workflow which is part of a larger workflow.
Password Intercept	Workflow activated when a password change interception event is received.

Sub-process Workflows

Some complex workflows are divided into multiple sub-process workflows that are activated by a master workflow. Using sub-process workflows with a master workflow can:

- Simplify the structure of the master workflow
- Make workflows easier to manage
- Promote re-usability because more than one master workflow can reference the same sub-processes

As a standard practice, these smaller workflows are assigned the **Subprocess** type of workflow. This type is not associated with any system functionality. However, using the Subprocess type designation enables you to easily identify the workflow as a sub-process of a larger workflow.

Transient Workflows

Transient workflows are launched in a special mode that does not persist any information to the database. A workflow remains in the transient state until the workflow reaches an approval step. If the workflow launches to completion without an approval step, nothing is stored in the database unless the browser terminates or the session times out the workflow and any progress made is lost.

Note: If the browser terminates or the session times out the workflow and any progress made is lost.

Examples of transient workflows include:

- QuickLaunch workflows that can present a series of forms before performing any relevant actions.
- Self-registration workflows that do not require authentication
- Workflows for users trying the registration process, who do not have an inbox where they can see their past attempts

To create a transient workflow, add a variable named **transient** and set the value to **true**.

For transient workflows to work correctly, the user interface code needs to manage the workflow case in a special way, through a WorkflowSession.

The case persists when any of these things happen:

- an approval for someone that is not the submitting user
- a step with a wait='x' in it
- a step with background='true'

Chapter 13: Using the Business Process Editor with Workflows

The IdentityIQ user interface provides a graphical tool for defining and editing workflow processes. You can use the IdentityIQ Business Process Editor to:

- Create a new workflow or edit an existing workflow
- Set up the workflow structure.
- Create the steps that define the behavior or the workflow.
- Outline the transitions between the steps.
- Define forms.
- Assign conditions.

This tool also provides a graphical representation of the process flow that can be used to create documentation about the activities included in the workflow.

Typically, administrators use the graphical editor to outline the process and then move to the XML representation to add to or adjust the details of each step. After you save the process, you can view, edit or export the XML representation from the IdentityIQ Debug pages.

Note: Because some workflow steps cannot be defined with the graphical editor, workflow development can involve direct editing in the XML representation and some amount of Java coding. An understanding of XML and Java syntax is a general requirement for workflow development.

Creating and Editing Workflows

Use the Business Process Editor to create a new workflow or edit an existing workflow. Original workflows can also be created from existing processes.

Basic Workflow How-To Tasks

You can perform the following tasks:

- "How To View or Edit a Workflow" on page 167
- "How To Create a New Workflow" on page 168
- "How To Use an Existing Workflow to Create a New Business Process" on page 168

How To View or Edit a Workflow

1. Navigate to **Setup** -> **Business Processes**.
2. Select an existing workflow from the **Edit an Existing Process** list.
3. Navigate through each of the process tabs to view or modify the workflow data.
4. To save changes to an existing workflow, click **Save**.

How To Create a New Workflow

1. Navigate to **Setup** -> **Business Processes**.
2. Click **New** to create a new workflow and then enter a name for your process.
3. Specify a name and description for the workflow. Use a short descriptive name for the workflow and use a the description that provides an overview of the workflow function.
4. In the **Type** field:
    a. Select from the drop-down list of predefined workflow types. The available types are restricted to the process options related to the workflow.
    b. To enter a custom type, manually enter the type name in the box instead of selecting one from the list. See the Workflow Basics chapters for any limitations to custom types.
5. Navigate through each of the process tabs and specify workflow data.
6. Click **Save**.

How To Use an Existing Workflow to Create a New Business Process

1. Navigate to **Setup** -> **Business Processes**.
2. Select an existing workflow from the **Edit an Existing Process** list.
3. Navigate through each of the process tabs to view or modify the workflow data.
4. Click **Save As** and enter a unique name for the workflow.

Process Editor Tabs

The Process Editor has the following tabs:

Table 52—Process Editor Interface Tabs

Interface Tab	Inputs
Process Details	Specify Name, Type, and Description of the workflow. See "Process Editor Tabs" on page 168.
Process Variables	Specify any variables that apply to the workflow. Variables in any input variables, return values, and working variables for use within the process's steps. See "Process Variables Tab" on page 170.
Process Designer	To graphically represent the process, specify the actions involved in each step, and provide the evaluation conditions for moving from one step to another. See "Process Designer Tab" on page 171.
Process Metrics	Review statistics gathered for the process as it launches. See "Process Metrics Tab" on page 181.

Process Details Tab

The Process Workflow tab has the following options:

Table 53—Process Detail Tab - Available Process Types

Process Type	Description
Policy Violation	Workflow activated to launch policy violation actions.
Batch Provisioning	Workflow activated to launch batch requests.
Scheduled Assignment	Workflow activated to when a role is ready to be assigned.
Scheduled Role Activation	Workflow activated when a role is ready to be enabled or disabled.
Managed Attribute	Workflow activated when an entitlement is created or edited.
Identity Correlation	Workflow activated when performing identity correlation tasks.
Identity Event	Workflow activated for identity event. For example, sunrise/sunset dates for deferred entitlement, role assignment, or role removal.
Identity Lifecycle	Workflow activated for Lifecycle events. For example, Lifecycle Event - Joiner or Lifecycle Event - Leaver.
Identity Update	Workflow activated when you update an identity through the **Define** -> **Identity** page. Typically requires few or no approvals.
Identity Refresh	Workflow activated for identities that are refreshed using the Identity Refresh task. This type of process can be used for additional customization during refresh, and to present provisioning policy forms if accounts need to be created as a result of automated role assignment.
LCM Identity	Workflow associated with Lifecycle Manager Identity related tasks, for example, LCM Create and Update.
LCM Provisioning	Workflow activated for Lifecycle Manager provisioning tasks.
LCM Registration	Workflow activated for registration tasks.
Policy Violation	Workflow activated to initiate policy violation actions.
Role Modeler	Workflow associated with Role functions. For example, Owner Approval and Role Activation.
Subprocess	Designation of a workflow which is part of a larger workflow.
Password Intercept	Workflow activated when a password change interception event is received.

Process Variables Tab

The Process Variables tab lists variables you can use with the workflow. For most of the default processes, the variables are listed in a collapsed, advanced view. You can expand the view to show the details for each variable. Variables include:

- Input variables for workflow
- Output variables for workflow
- Working variables used for processing a workflow

Variables are marked as **Input**, **Required**, **Editable**, or **Output**.

To delete a variable, expand the variable and click **Remove**.

Table 54—Process Variable Flags

Object	Usage
Input	Specifies that the variable is one of the arguments to the workflow, passed in when it is launched.
Output	Stores the variable in the workflow's task result to allow the user to view the progress and results of the workflow. To view the results, navigate to **Setup** -> **Tasks** -> **Task Results**.
Editable	Enables the variable to be edited in the basic view.
Required	Indicates that the variable must contain a value (non-null) when the workflow starts.
Description	A brief description of the variable and its function.

Note: The order of variable declarations can make a difference. For variables in the XML that reference other variables in their initializations, the referenced variable must be declared first.

When variables are created through the user interface, the new variables are inserted in the list above the existing variables. When the XML representation of the workflow is generated, the variables are listed in the order they were created, which is the opposite of the display order in the user interface.

Basic View

IdentityIQ has several built-in business processes that are available when you install the product. The commonly used processes are available through the Basic View which is a simplified, form-based view. The information you edit in the Basic View can be also be configured or removed using the Advanced View. The Basic View includes the following business processes:

Note: Business processes with the LCM label are part of IdentityIQ Lifecycle Manager, which is licensed separately.

- Identity Update
- LCM Create and Update
- LCM Manage Passwords
- LCM Provisioning
- LCM Registration

How to Use the Basic View

1. Navigate to the Debug page and edit the XML of the business process.

2. Manually add and configure the **configForm** attribute to reference the form to be presented in the Basic section of process variables. See also “Editing Workflow XML” on page 183.

   Note: If the reference exists in the business process, but the form does not, an error is displayed and you are returned to the Advanced view.

Variable Initialization

To initialize variables for the workflow, specify an initial value for the variable in this panel. For best results, use initial values for the workflow variables, rather than creating multiple process steps to initialize each variable.

There are five ways that initialization can occur:

Table 55—Variable Initialization Options

Object	Usage
String	Assigns a literal value to the variable.
Reference	Sets the variable value through a reference to one of the other workflow process variables.
Script	Sets the variable with a Beanshell script inside the workflow.
Rule	Sets the variable by calling a Beanshell rule outside the workflow.
Call Method	Assigns the return value of a call to a compiled Java method in a workflow library to the variable.

Note: Variable values passed into the workflow through workflow arguments supersede variable initial values. Therefore, any value provided in an argument overwrites the initial value for that argument.

Timing of Variable Definition

Variables that are known at the beginning of workflow development can be defined before the graphical process design begins. Throughout the development process you might need to define other variables. Variables are not restricted to only those that were previously defined on the Process Variables tab. Variable definition can be done before, during, or after the design process.

Process Designer Tab

The majority of the work in creating and modifying a workflow is done on the Process Designer tab. The steps and transitions you create for workflow determine the workflow activities and can include the following items:

- “Process Steps” on page 171
- “Approval Steps” on page 175
- “Form Steps” on page 177
- “Step Conditions” on page 180
- “Step Transitions” on page 180

Process Steps

A workflow involves a minimum of three steps: a start step, a processing step, and a stop step or END. For best results, all workflows should contain a start and stop step and that these two steps contain no actions. Workflows can contain as many or as few processing steps as are necessary to manage the required actions. To add steps using the Process Designer, navigate to the Process Editor and click the desired step type in the **Add A Step** section. You can drag steps around the Process Designer grid to line them up visually in a logical progression.

To add a new step:

1. Click **Add a Step** in the left-hand column to display panel that contains available steps.

   Note: Only steps associated with the process type and that exist in the Step Library are listed in the Add a Step panel.

2. Click and drag the desired step to a position in the process design grid.

To edit to the contents of a step,

1. Right-click the step icon and select **Edit Step**.
2. The step details window displays. You can:
   - Record the **Name** and **Description** of the step.
   - Name the **Result Variable**, a variable to receive the resulting value of the step action.
   - Specify the **Action** for the step.

Each step can take one of the types of actions listed in the following table.

Table 56—Process Step Action Types

Object	Usage
Script	Executes a segment of Java BeanShell code that is included in the step.
Rule	Executes a workflow Rule — a block of Java BeanShell code encapsulated in a reusable rule.
Subprocess	Launches another defined workflow, passing control to it until it completes. When you select this option, the list of available subprocesses, workflows of type Subprocess, displays and you are given the option to enable step replication.
Call Method	Calls a compiled java method in the IdentityIQ workflow library, exposed through the standard workflow handler. When you select this option, the list of available methods displays.

Note: For any of these actions, an appropriate value must be specified or selected before the action can be saved.

For example, if Script is selected, a BeanShell script must be entered in the box. If you choose the Subprocess object, a subprocess must be selected from the list. If the value is not specified, the step is saved with no associated action. Developers who use subprocesses must write the subprocesses before they can complete the step definition of steps in the master process.

The **Enable Monitoring** flag on this window turns on metrics tracking for the step. See “Process Metrics Tab” on page 181 for more information on process monitoring and metrics.

Script

Scripts are java BeanShell code that you write in order to execute a desired action. You write scripts directly in the **Source** box in the detail window for the step.

Note: The script examples in this document all show very short java BeanShell code blocks. There is no set length for a script. A script block within the XML can be any length needed to accomplish the required processing. However, long scripts are frequently encapsulated in rules, as discussed in the next section.

Rule

Rules are also blocks of java BeanShell code. Code encapsulated in a rule is available for reuse by other areas of the application that can launch a rule of the same type. Rules created through this window are of type Workflow and can be used by any workflow. When you choose **Rule** as the **Action**, you can select an existing workflow rule from the list or create a new rule in the rule editor. To open the rule editor, Click the **. . .** icon.

Subprocess

Subprocesses are other workflows. You can use subprocesses to subdivide complex processes into smaller segments that can be easily managed and reused by other workflows. Subprocesses are complete workflows that contain a start step, a stop step, and as many processing steps as are needed to complete their activities.

You can enable step replication to enable multiple subprocesses to run to completion at the same time instead of having them run serially. For example, in an approval step, you can launch multiple approval subprocesses, to multiple approvers, that can take an approval all of the way through provisioning instead of the approval step waiting for all approvals to complete before provisioning can begin.

When you enable replication, you must select an item from the main workflow for replication and an argument that is passed to the action containing the replicated item. Only one item can be replicated per step, and all of the items must be passed the same argument. A new subprocess is generated for each item replicated.

Call

The IdentityIQ workflow library contains a set of methods that you launch within a workflow. Methods are exposed through the standard workflow handler that the workflow engine calls every time an action occurs in a workflow. Every workflow has access to the methods in the standard workflow handler. Additional libraries of methods are also available to use in workflows.

Note: When no library list is specified for the workflow, the default includes access to the Identity, Role, PolicyViolation, and LCM libraries.

Through the XML, you can specify other libraries, including custom libraries for an installation. The user interface does not provide an option to manage the library list

Note: Specifying a library list overrides the default. You must explicitly include in the library list any default libraries that contain methods the workflow needs. See “Workflow Element” on page 185 for more details on specifying a library list.

When **Call Method** is selected for the workflow step, **Action**, the method name is selected from the **Call Method** list. The methods in these workflow libraries are listed and briefly described in “Workflow Library Methods” on page 206.

Step Arguments

When arguments need to be passed to the script, rule, subprocess, or library method launched by a step, you must specify the argument on **Arguments** tab for the step. Arguments can be specified in the following ways:

Table 57—Step Argument Specifications

Type	Usage
Basic View	Some steps copied from the step library include a configuration form to simplify the specification of arguments. When a step has a configuration form, this is called the Basic View and is shown by default. The Basic View allows you to set arguments using literal values.
Advanced View	The advanced view gives you more control over how the argument values is calculated.
Return Variables	Each step can return only one result variable, which can be specified through the user interface. When a step has an action that launches a subprocess, you can also use return variables. Multiple values can then be passed back from the subprocess to the main workflow. Because the user interface does not provide a vehicle for declaring return variables, You must specify the return variables directly in the XML. See “Return Variables” on page 170 for more information.

Table 58—Step Argument Specifications

Type	Usage
String	A literal value. For example, the name of an email template to use.
Reference	A reference to one of the workflow's process variables.
Script	A segment of java BeanShell code that returns a value.
Rule	A workflow rule that returns a value. This functions similar to Script except BeanShell is contained within a re-usable rule.
Call Method	A call to a workflow library method that returns a value.

When a script, rule, or library method is used to calculate an argument value, the configuration can be more complex. If the argument definition needs data to be passed in, you can pass the data by:

- Providing all the current values of workflow variables.
  OR
- Declaring the value of step arguments above an argument.

If desired, you can use ordered step arguments instead of workflow variables if the only use for the value is within this step.

For example, when these two step arguments are declared in this order, the method called to populate Identity_mgr can use the value in Identity_name in its processing if needed.

Table 59—Step Argument Example

Argument Name	Value Type	Value Source
Identity_name	Reference	IdentityName
Identity_mgr	Rule	getManagerRule

More on Start and Stop Steps

Similar to other steps, start and stop steps can contain actions that launch scripts, rules, subprocesses, or calls to workflow library methods. By convention, these steps are included in every workflow but are used only to designate a clear starting and ending point for the workflow. These steps are generally empty steps with no action. Occasionally, debugging messages can be printed from these steps to trace workflow progress during development.

Step Icons

When steps are first added through the Process Designer, only three icon types are available: Start, Stop, and Generic Step. A variety of other icons are available. You can use different icons to make it easier to determine the actions each step performs.

To change an icon for a step:

1. Right-click the step icon and click **Change Icon**.
2. Select the desired icon style from the pop-up window.

Approval Steps

Approval steps are a special type of step in IdentityIQ. You can use Approvals to gather data from a user through a work item. In an approval, the user is asked to review a requested action, such as, granting a role to an identity, and then give their approval for the action to be processed.

To create a basic approval through the user interface:

1. Right-click the step.
2. Click **Add Approval**.

   Note: A step can contain an action or an approval, but not both. Approval steps are used for approval processing. Approval steps are not used to perform other actions such as scripts, subprocesses, etc.

To edit an approval that exists in a step

1. Right-click the step and click **Edit Approval**.
2. Alternatively, you can choose **Edit Approval** from the Step Details window.

Approvals are flexible and meet a variety of business needs. An approval can be constructed many ways. approvals range from a simple one-person approval to a complex approval process that involves multiple people with different approval modes and notification schemes.

Approval Details

Every approval includes the following fields to be completed on the Details tab for the approval:

Table 60—Approval Step Details

Object	Usage
Name	User-defined name for the approval.
Send	Comma-separated list of process variable names to be sent to the approval.
Return	Comma-separated list of variables names to copy from the completed approval work item back into the workflow.

Table 60—Approval Step Details

Object	Usage
Renderer	JSF (Java Server Faces) include to render the work item details. Not required if using a default renderer.
Mode	Specifies how approval is processed when multiple owners are specified.
Owner	Approver for the approval. Can be more than one Identity name and is specified by string, reference, script, rule, call method. When more than one owner is specified, mode determines how and when the item is submitted to each listed owner. Parallel, parallelPoll, and any modes submit the approval work item to all owners at the same time. Serial and serialPoll modes wait until the first owner completes the approval before submitting to the next approver in the list.
Description	Defines work item description. Shown as the work item Name in the approver's inbox. Set using string, reference, script, rule, call method.

Approval Arguments.

You can set arguments to the approval on the **Arguments** tab. Generally, variables are passed to approval through the send list. However, any arguments that require transformation, through script, rule, or library method, must be sent through an Arg element. Args defined with reserved system names are passed through the Arg element with the reserved name specified. See “Approval Steps” on page 200 for information on reserved system names.

Work Item Configuration

You can specify some details about the notification and escalation/reminder policy for a work item on the **Work Item Configuration** tab. The work item appears in the owner's IdentityIQ inbox and requires their input. If no configuration is specified, the default work item configuration is used.

To change the configuration for the work item

1. Select **Override Work Item Configuration.**
2. To include an electronic signature in the approval step, select **Override Electronic Signature Configuration**.

The following configuration options are available on the **Work Item Configuration** tab:

Table 61—Work Item Configuration Options

Option	Description
Initial Notification Email	To change the notification email template, select the template from the list

Table 61—Work Item Configuration Options

Option	Description
Escalation	Choose an escalation policy: **None**: no escalation**.** **Send Reminders**: allows configuration of reminder options, such as days before first reminder, frequency, email template**.** **Reminders then Escalation**: allows reminder option configuration plus escalation option configuration, such as reminders before escalation, escalation owner rule, escalation email**.** **Escalation Only**: allows configuration of escalation options, such as days before expiration, escalation owner rule, escalation email).

Child Approvals

Use Child Approvals to customize approval processing or presentation for the different sets of identities involved in the approval process. For example, a change in a user's assigned region requires someone in HR sign off and also requires manager approval. Although the approval of the user's own manager is required, any HR individual can completes the sign-off. This type of approval can be created through child approvals.

To create a child approval:

1. Click **Add Child Approval** on the **Details** tab for the parent approval.
2. Click the child approval in the **Approval Children** hierarchy to select it for editing.

To set up the approval described in the example, create two child approvals:

- HR Approval set up — any of the identities who meet the criteria can make the decision for the group
- Manager Approval set up — identity's manager specified as the owner.

Note: The reference variables HRApprovers and identityManager for the example are process variables defined with initialization scripts that retrieve the appropriate sets of Identities.

If either approval requires a custom work item configuration, you can specify the configuration on the **Work Item Configuration** tab for the approval. Work item configurations are inherited by child approvals if configurations are not specifically overridden for the child. If you want a single custom work item configuration for the entire set of approvals, the configuration should be specified on **Work Item Configuration** tab for the parent approval. In this case, the child approvals inherit the parent configuration.

Form Steps

An approval step can also display a form. Forms are a general way to request information from the user and do not necessarily represent an approval. For example, you can use forms to request a missing attribute such as the department name for an identity or ask the requester for more information about why they are making the request.

You can define a form inside the workflow step or you can reference an external form that is shared with other workflows.

To reference an existing form:

1. Right-click the step and click **Add Form**.

2. In the first screen, click **Reference Form**.
3. In the form reference screen, select a form from the table and select the owner who will be shown the form.

To create a custom form for gathering data from a user:

1. Right-click the step and click **Add Form**.
2. In the first screen, click **Create**.
3. In the form editor, specify the general form properties.

Table 62— Form Step Properties

Field	Description
Description	Work item description text displayed on the user's Home Page.
Send	Comma-separated list of process variables to be passed as initial values for the form fields.
Return	Comma-separated list of form fields to copy back into process variables when the work item is closed.
Owner	The identity to be shown the form. Can be a simple identity name, a name stored in a process variable, or a name calculated by a script, rule, or library method.

A form includes one or more <i>fields</i> that define what information you want to show and the information you are asking the user to provide. This form field editor is similar to the field editor for provisioning policies and uses most of the same options.

Table 63— Form Step Values - Bottom

Field Attribute	Description
Name	System-accessible name for field. Used to reference field programmatically.
Display Name	Label that is displayed on form for the field.
Help Text	Tool tip help text for field.
Type	Field type. Impacts rendering of field on form.
Multi-valued	Flag to determine if the field can contain multiple values (multi-selectable).
Read Only	Field displays a value that cannot be changed.
Hidden	Field is not displayed.
Owner	Field owner. Does not apply to form fields.
Required	Value must be entered.
Refresh Form on Change	Form is refreshed when the value for this field is changed. *TIP! This field is useful when the value of a field in the form depends on the value in another field.*
Display Only	Does not apply for workflow forms.
Authoritative	Does not apply for workflow forms.
Value	Literal, script, or rule to set the initial value of the field.

Table 63— Form Step Values - Bottom

Field Attribute	Description
Allowed Values	Allowed values for the field. Displays as a drop-down list box or combo box based on the multi-valued setting.
Validation	Rule or script that validates the value of a field when the form is saved/submitted. Prevents submission if the value is not valid.
Dynamic	Delays the launch of allowed values, scripts, or rules until the field is selected, instead of launching as soon as the form loads.

The form editor also provides the option to specify buttons to include on the form.

To add a button definition:

1. Click **Add Button**.
2. Select the button **Action** and specify a behavior of the button.
3. Specify addition button options as described in the table below, and click **Save**.

Table 64— Button Properties

Function	Description
Action	Select the action the button takes when pressed. Choose from the following actions: **Next** — assimilates form data and advances to the next state, such as OK/Save/Approve/Submit functionality. Sets status of approval to Approved. **Cancel** — Stops form editing, returns to previous page in the user interface, and leaves work item active. **Back** — assimilates form data and returns to the previous state. Sets status of approval as **Rejected** and advances workflows. **Refresh** — Assimilates the posted form data and regenerates the form.Not a state transition. Refresh is a re-display of the form.
Label	Text to display on the button.
Parameter	Name of an optional value to be sent with the form fields when this button is pressed.
Read Only	Non-actionable button.
Skip Validation	Ignores the validation when the form is posted.
Value	Optional value to be sent with the form fields when this button is pressed.

During initial form specification, defined buttons and fields are listed together in the left panel in the order they are added. If some buttons were added before some fields, the button can be intermixed. On the final form, buttons are always grouped together at the end of the form. When the Form Editor is revisited later, the fields are listed together first, in the order they were created, and then the buttons follow in the order they were created.

Note: Buttons can be reordered in the XML to display in a different order on the form.

Custom forms can also be created or edited through XML. Various advanced form options, such as sections, multi-column layout, are only available through the XML. See, “Forms” on page 23 for more information.

Step Conditions

Normally when a transition is made into a step, the step action is executed. In some cases you might want the execution of the step to be optional. You can add a step condition to control whether or not the step action executes. Step conditions can also simplify transition lines in the process because you do not have to create many complex transitions to skip over steps. You can advance from one step to another and let the step conditions determine if the step is executed.

To edit the step conditions:

1. Right-click any process step.
2. Click **Add Step Condition**.
3. Specify addition button options as described in the table below, and click **Save**.

You can express conditions as any of the following:

Table 65—Step Transition Conditions

Type	Description
Reference	Evaluation of a defined process variable. Must be a Boolean variable.
Script	Segment of java code that evaluates process variables.
Rule	Workflow rule that contains a reusable segment of java code to evaluate process variables.
Call Method	Call to launch a Java method in the IdentityIQ workflow library. Exposed through standard workflow handler.

Selecting the **Negate** option changes the evaluation to the opposite condition. For example, if the condition evaluates to False, the negate option changes it to True.

Step Transitions

Steps are connected through Transitions. Transitions can connect one step to the next sequentially. Alternatively, steps can include evaluation statements that enable conditional processing, such as certain data conditions that can cause the workflow to execute Step A versus Step B.

To add a transition do the following:

1. Right-click the process step for starting the transition and select **Start Transition**.
2. Navigate to the process step for ending the transition, right-click and select **End Transition**.
3. Right-click the transition icon and select **Edit Transition** to set the condition.
4. To add additional conditions to this transition, repeat the process.

To edit the transition conditions:

1. Right-click the transition diamond
2. Click **Edit Transitions**.
3. Specify addition button options as described in the table below, and click **Save**.

A step can have as many transitions to next steps as needed. Transition conditions are evaluated in the order they are listed. The first transition that has no condition, or whose condition evaluates to true is taken. Use the up and down arrows in the transitions dialog box to re-order the transitions. As a recommended practice, the final transition should have no condition. That transition is taken when no other transition conditions are met. If a step only has transitions with conditions, and none of the conditions are met, the workflow ends.

Conditions can be expressed as any of the following:

Table 66—Step Transition Conditions

Type	Description
String	Not used. This condition is an artifact of the common structure used for variable setting and does not apply to conditions. A literal value of **True** or **False** can be specified but does not allow any evaluation in the transition. **True** always launches the associated step and **False** always bypasses the associated step.
Reference	Evaluation of a defined process variable. Must be a Boolean variable.
Script	Segment of java code that evaluates process variables.
Rule	Workflow rule containing reusable segment of java code to evaluate process variables.
Call Method	Call to launch a Java method in the IdentityIQ workflow library. Exposed through standard workflow handler.

Transition conditions must evaluate to boolean values. If the value is true, the workflow moves to the step that the transition references. If the value is false, the next transition in the list is evaluated.

Selecting the **Negate** option changes the evaluation to the opposite condition. For example, if the condition evaluates to False, the negate option changes it to True.

Note: Because all processing options should end with the stop step, every workflow should end with a step that transitions to Stop.

Process Metrics Tab

The Process Metrics tab displays the following statistics that are useful for troubleshooting workflows:

- Number of times the workflow launched.
- Number of times the workflow succeeded or failed.
- Average and maximum duration of the workflow.
- Date the workflow last launched.

You can view additional process metrics, including data tracked at the step level, through the **Intelligence -> Advanced Analytics** -> **Process Metrics Search** tab.

To turn on metrics tracking:

1. For individual workflow steps, select **Enable Monitoring** in the Details window.
2. Alternatively, you can right-click on a step and select **Enable/Disable** from the drop-down menu on the step.

To turn on monitoring for all steps in a workflow, click **Monitor** at the bottom of the business process editor window.

Chapter 14: Editing Workflow XML

There are various options for editing workflow XML. You can:

- Create the initial workflow through the user interface and then edit the workflow directly.
- Complete all workflow development in XML.
- Write original XM or use XML from an existing workflow as a template for a new process

All of these methods are valid and can be used as desired.

Accessing the XML

The XML for existing workflows can be viewed and edited through the IdentityIQ Debug pages or can be exported through the IdentityIQ Console.

Debug Pages

To view the XML in the Debug pages, navigate to the Debug pages and Select **Workflow** from the object list to view a list of all defined workflows in the system.

to view the XML representation, click the name of the workflow. From the Debug pages you can edit and save changes. A workflow can also be copied from here and pasted into an external editor of choice.

- View and edit the XML.
- Save changes to the XML.
- Copy and paste the XML to an external editor.

IdentityIQ Console

You can export one or more workflows from IdentityIQ through the console. The console export is the most efficient way to get the XML for all workflows extracted from the system at one time. The IdentityIQ console export command can extract all the Workflow XMLs together into a single file.

After export the XML, you can parse the XML into a separate file for each workflow and save the files in the installation source code control system for later use in system environment migrations or in product upgrade processes.

Table 67—Important Workflow Objects

Object	Usage
Workflow	Defines the workflow structure and steps involved in the workflow processing.
WorkflowCase	Represents a workflow in progress. Contains a Workflow element in which the process is outlined and current state data is tracked, as well as identifying information about the workflow target object.

Table 67—Important Workflow Objects

Object	Usage
WorkflowContext	launchtime information that Workflower maintains as it advances through a workflow case. Passed into rules and scripts and to the registered WorkflowHandler. Contains all workflow variables, step arguments, current step or approval, workflow definition, libraries, and workflowCase.
Task Result	Records the completion status of a task, or in this case, the workflow, contained within the WorkflowCase.

Re-importing the XML

Because the system only launches Workflow XML that is saved within IdentityIQ, XML documents that are edited externally must be re-imported for the changes made to them to take effect.

To re-import an externally saved XML document, use the console import command or from the Import from File page accessed from the Global Settings page.

Dollar-Sign Reference Syntax

You can reference workflow variables inside XML tags and in user interface fields using `$()` notation. These are resolved into their variable values. For example, if a variable identityName is defined and contains the full name of an Identity, for example, John Smith, an Arg specified as:

```
<Arg name="FullIdentityName" value="$(identityName)">
```

passes “John Smith” as the value for the variable **FullIdentityName**.

When the variable is used alone, it functions the same as specifying value="ref:identityName. However, the more common usage is to include the variable in a longer string such as:

```
<Arg name="Title" value="Role Update for $(identityName)">
```

which passes “Role Update for John Smith” as the value for the variable Title.

XML Content

This section describes the elements present in the workflow XML and explains their usage.

Header Elements

The following three lines must be included as shown in any workflow document. The `<sailpoint>` tag must, of course, be matched with a `</sailpoint>` tag at the end of the workflow document.

```
<?xml version='1.0' encoding='UTF-8'?>

<!DOCTYPE SailPoint PUBLIC "sailpoint.dtd" "sailpoint.dtd">

<sailpoint>
```

Workflow Element

The Workflow tag identifies the name and type of the workflow.

```
<Workflow  explicitTransitions="true" name="WF-Training Hello World Workflow"
type="IdentityUpdate">
```

The attributes of a workflow element including the following:

Table 68—Workflow Element Attributes

Workflow Attribute	Purpose
configForm	A soft reference to a process variable form presented in the Basic View of Process Variables tab, or a step form presented in the Basic View of the Arguments tab on the Step Editor panel accessed from the Process Designer tab of the Manage Business Process page.
name	Short descriptive name for the workflow this is displayed in user interface selection list-boxes and list of existing business processes on the Process Editor window.
type	Workflow type. Type is used to filter workflow selection lists in configuration windows where you select the workflow based on system activities.
explicitTransitions	Boolean value indicating that transitions between steps are explicitly specified and workflow should not resort to implicit, fall-through, transitions when no transition conditions evaluate to **true**. The default setting is **false.** If you so omit this argument and the specified transition conditions all evaluate to **false**, the workflow uses implicit transitions and launches the next sequential step in the XML. However, if you edit a workflow using the Business Process Editor, the value is changed to **true**. **Note: If the developer makes the last transition in any set unconditional, which is considered best practice, the transitions between steps are smoother.**
libraries	Lists workflow libraries the workflow needs. **Note: If this attribute is not specified, workflows automatically have access to Identity, Role, PolicyViolation, and LCM libraries.**
stepLibraries	Lists workflow step libraries the workflow can access. **Note: If this attribute is not specified, workflows automatically have access to the Generic Step Library, which provides access to the Start, Stop and Generic steps.**
handler	The default workflow handler is sailpoint.api.StandardWorkflowHandler. This attribute does not need to be specified when the default is used. In this case, the best practice is to omit it. **Note: If you use a custom workflow handler, the custom handler must EXTEND the default handler and not replace it. The custom handler must be specified in the workflow Handler argument.**

Variable Definitions

The recommended best practice is to identify all variables for the workflow at the top of the XML document. The variable definitions come next in the XML.

At a minimum, variable elements require a name. Other attributes can indicate the variable type and use, such as input, required, editable, return. A description can be specified for each variable. When needed, an initialization value can also be provided. Using the initialization option is the recommended practice rather than creating separate steps to initialize each variable. Using initialization values is more efficient, easier to read, and easier to debug, because Trace reports initializations as they occur. For more information, see "Initializer Options" on page 187.

```
<Variable input="true" name="project" output="true" required="true">
    <Description>
      Project that has account requests in the QUEUED state.
    </Description>
  </Variable>


<Variable editable="true" initializer="true" name="doProvisioning">
    <Description>Set to true to cause immediate provisioning after the
assignment</Description>
  </Variable>
```

Some parts of the variable definition are expressed within attributes on the Variables element. Other parts are expressed through nested elements of their own.

Table 69—Variable Attributes

Variable Attribute	Purpose
name	Variable name.
type	Variable type. Type declaration is not enforced by the application and is used primarily for documentation.
initializer	Initialization value for the field.
input	Flag indicating that the variable is an argument to the workflow. Omitted if not true.
output	Flag indicating that the variable is a return value for the workflow. Omitted if not true.
required	Flag indicating that the variable is a required field for the workflow. Omitted if not true.
editable	Flag indicating that the variable can be edited by the workflow. Omitted if not true.
Nested Tag within Variable Element	
Description	Provides a description of the purpose for the variable.
Script	Alternative to script in the initializer attribute value. Should be used for initializer scripts of any length or complexity.
Source	Nested within the Script tag and contains the java BeanShell source for the action to be executed.

Initializer Options

The Initializer attribute requires additional attention. When these attributes are set through the user interface, you can specify the attribute as a string, script, rule, call, or reference. The same options are available directly through the XML.

Note: The initializer for a variable is only used when a value for the variable is not passed in to the workflow.

Table 70—Initializer Options

Initializer Type	Description and Examples

string	Assigns a literal value to the variable.
	
	Note: String is the default initializer option so the "string:" prefix can be included or omitted.
	
	Examples:
	`<Variable initializer="string:true" name="trace"/>`
	
	`<Variable initializer="spadmin" input="true" name="fallbackApprover">`

Table 70—Initializer Options

Initializer Type	Description and Examples

script	Assigns a value based on the results of a Java BeanShell script.
	
	Examples:
	
	(1) In-line Script. Only use for very short, simple scripts.
	
	`<Variable initializer="script:(identityDisplayName != void) ? identityDisplayName : resolveDisplayName(identityName)" input="true" name="identityDisplayName">`
	
	(2) Script within nested <script> element. Use for most script initializers - scripts of any complexity or length.
	
	`<Variable initializer="script:resolveDisplayName(launcher)" input="true" name="launcherDisplayName">`
	`   <Description>`
	`      The displayName of the identity being who started this workflow.`
	`      Query for this using a projection query and fall back to the name.`
	`   </Description>`
	`   <Script>`
	`      <Source>`
	`         // Lookup the launcher's display name for use in email templates.`
	`         String returnString = launcher;`
	`        Identity launcherId  = context.getObject(Identity.class, launcher);`
	`         if (null != launcherId) {`
	`            returnString = launcherId.getDisplayName(); // First+Last`
	`         }`
	`         return returnString;`
	`      </Source>`
	`   </Script>`
	`</Variable>`

rule	Assigns a value based on the return value of a workflow Rule. Examples: `<Variable initializer="rule:wfrule_GetIdentityName" name="IdentityName">`
call	Assigns a value based on the return value of a call to a workflow library method. Example: `<Variable initializer="call:getObjectName" name="roleName">`

Table 70—Initializer Options

Initializer Type	Description and Examples
ref	Assigns a value based on a reference to another workflow variable. This type is rarely used. Example: `<Variable initializer="ref:otherVar" name="myVar"/>`

Workflow Description

A description element should be included to describe the purpose of the workflow. Although the description element is not used in the workflow process, it is recommended for usability. In the user interface, the contents of this element are displayed on the Process Details tab of the Business Process page. This element should be included near the top of the workflow, either before or after the variable definition section.

```
<Description>
     Workflow called when a role is ready to be enabled.
</Description>
```

Rule Libraries

Some methods the workflows use are grouped together into Rule Libraries. These Rule Libraries are defined as rules in IdentityIQ. However, these libraries contain sets of related but unconnected methods that workflow steps can directly within a script action. Because the rule methods are in rules, rather than in the compiled Java classes, their functionality can be easily modified to meet the needs of each installation. To make the methods within one of these rules available to steps within the workflow, the RuleLibraries element must be declared. See the following example.

Note: Each Reference element applies to one library. Include only the libraries that contain the required methods in the RuleLibraries declaration for the workflow.

```
<RuleLibraries>
    <Reference class="sailpoint.object.Rule" name="Workflow Library"/>
    <Reference class="sailpoint.object.Rule" name="Approval Library"/>
    <Reference class="sailpoint.object.Rule" name="LCM Workflow Library"/>
</RuleLibraries>
```

Note: You can create and reference custom libraries using this same syntax.

Step Libraries

Step libraries are designed to offer a group of common functions that can be added to existing workflows from the **Add a Step** panel Business Process Editor. Step libraries are a collection of steps encapsulated by a workflow with the template attribute marked true. The steps do not have any transitions and they are not executable. A Step Library must be defined. See the following example.

Note: The type does not have to be StepLibrary. However, using the StepLibrary type ensures that these workflows do not appear in other parts of the product.

```
<Workflow name="Provisioning Step Library"
                  type="StepLibrary"
                  template="true">
```

When you edit a new or existing workflow, you can include a list of step libraries by including a comma separated list in the stepLibraries attribute. See the following example.

```
<Workflow name="LCM Provisioning"
                  type="Provisioning"
                  taskType="LCM"
                  libraries="Identity,Role,PolicyViolation,LCM,BatchRequest"
                  stepLibraries="Common,Provisioning"
                  handler='iiq.api.StandardWorkflowHandler'>
```

In the example above, when you edit a business process with the LCMProvisioning type, the Common and Provisioning steps are available in the **Add a Step** panel of the Business Process Editor.

Steps within a step library workflow can also include a soft reference to a step form that provides a simplified form-based interface that you can use to add arguments to some steps in the workflow. This form-based interface adds a Basic view option to the **Arguments** tab of the Step Editor. The Basic view is built using the information contained in the referenced form. The Advanced view is a list of all possible arguments and is built using the list of arguments that the step library references.

When you add a step form reference to a step library, use the configForm attribute, See the following example.

```
<Workflow name="Provisioning Step Library"
                 template="true"
                 type="StepLibrary">

 <Step configForm="Provisioning Approval Step Form"
                 icon="Task"
                 name="Account Approval">
<Arg name="approvalMode"/>

<Arg name="approvalScheme"/>

<Arg name="approvalSet" value="ref:approvalSet"/>
...
```

In the example above, when you edit an approval step in the Step Editor, the Basic and Advance Views of the Arguments tab are displayed.

Built-in Steps

IdentityIQ includes several built-in steps. The **Start**, **Stop**, and **Generic** steps apply to all workflow types. The following table lists the names, descriptions, and associated workflow types of additional built-in steps.

Table 71—Included Business Process Steps

Step	Description	Process Type
Notify	Allows users to select categories of recipients to notify, the specific recipient, recipients for each category, and the specific email template to use for each category.	Identity Lifecycle LCM Provisioning
Account Approval	Used for provisioning request approvals. The process assumes many of the Provisioning Workflow structures exist.	Identity Lifecycle LCM Provisioning

Step Elements

The core of the workflow is contained within the step elements. At a minimum, a step should contain:. The action attribute determines what processing the step performs. Steps usually contain one or more nested <Transition> elements and ideally also contain a nested <Description> element that tells the reader what the step is intended to do.

- an icon
- name
- posX attribute
- posY attribute

The action attribute determines what processing the step performs. Steps usually contain one or more nested <Transition> elements and ideally also contain a nested <Description> element that tells the reader what the step is intended to do.

```
<Step icon="Start" name="Start" posX="250" posY="126">
    <Description>
        The workflow's processing starts with this step.
    </Description>
    <Transition to="Initialize"/>
  </Step>
```

Note: Similar to variables, some parts of a step definition are included as attributes of the step and others are expressed as nested elements within the step.

Table 72—Step Element Attributes

Step Attribute	Purpose
configForm	A soft reference to the form that is presented to the Basic View of the Arguments tab on the Step Editor panel.
name	Short but descriptive name for step displayed in user interface graphical display below the step icon.

Table 72—Step Element Attributes

Step Attribute	Purpose
icon	Icon to display for the step in the user interface graphical Process Designer. Valid icon values include: Start, Stop, Default (Generic Step), Analysis (Launch Impact Analysis), Approval, Audit, Catches, Email, Message (Add Message), Provision, Task (Launch Task), and Und
posX, posY	X and Y indicate positions where the step icon should be displayed on the user interface graphical Process Designer grid. **Note: If you omit the posX and posY values, the icon is displayed at the top right of the grid. You can drag the icon around to create the desired layout at a later time.**
action	The processing action to take for the step, such as a script, rule, subprocess, or call. See “Step Actions” on page 195.
wait	Pauses the action for a specified duration, see “Wait Attribute” on page 199.
catches	Causes the step to be launch when **Complete** status is caught, rather than through a transition from another step. See “Catches Attribute” on page 199.
resultVariable	Variable name that contains the return value from the step.
Nested Tag within Step Element	
Description	Provide a description of the step purpose.
Transition	Identifies the next step the process moves to when the current step is complete, see “Transition Element” on page 192.
Arg	Passes variables to the step. Used for steps that require data to be passed in to them.
Return	Receives return values from subprocess steps, see “Return Elements” on page 197.
Script	Alternative to script in the Action attribute for the step. Use these step attributes for action scripts of any length or complexity.
Source	Nested within the Script tag and contains the java BeanShell source for the action to execute.

Transition Element

The transition element indicates the name of the next step the process executes following completion of the current step and is always nested within a step in the model. Transitions can contain conditions based on a string, script, rule, call method, or reference (similar to a variable initialization). The return value for conditions must be a Boolean (True/False). When multiple transitions are stipulated, they are evaluated in the order they are listed, and the transition for the first condition met is followed. The last transition in the list should, as a best practice, not contain any conditions so it can be used as the default action.

Transitions contain two attributes:

- to — next step
- when — condition for progressing to the next step

When a script is evaluated as the condition for a transition, it is often specified through these nested elements instead of as a **when** attribute on the transition element, especially if you use a long script.

Table 73— Nested Tag Within Transition Element

Nested Tag Within Transition Element	Purpose
Script	Alternative to script in the transition **when** attribute. The script should be used for scripts of any length or complexity.
Source	Nested within the Script tag. This tag contains the BeanShell source for the condition evaluation.

Example:

```
<Transition to=”end”>
    <Script>
       <Source>
           (“cancel”.equals(violationReviewDecision) || ((size(policyViolations)
            > 0 ) &amp;&amp; (policyScheme.equals(“fail”))))
       </Source>
    </Script>
</Transition>
```

Conditions in the **when** attribute can be specified using the following types of conditions:

Table 74—Transition Element Conditions

Condition Type	Description and Examples
string	Not used.This condition type is an artifact of the common structure used for variable setting and does not apply to conditions. A literal value of **True** or **False** can be specified. However, using one of those literal values does not enable any evaluation in the transition. **True** always executes the associated step and **False** always bypasses the step.

Table 74—Transition Element Conditions

Condition Type	Description and Examples
script	Evaluates script result value to determine step transition. Very short scripts are specified inline on the transition element, within the **when** attribute. Longer scripts are expressed within nested <script> and <source> elements. **Note: Because script is the default transition when option, the “script:" prefix can be included or omitted.** Examples: (1) In-line Script. Use only for very short, simple scripts. `<Transition to="Exit On Policy Violation" when="script:((size(policyViolations)> 0) &amp;&amp; (policyScheme.equals(&quot;fail&quot;)))"/>` (2) Longer script within nested <script> tag should be use for transition scripts of any complexity or length. `<Transition to="end"> <Script> <Source> ("cancel".equals(violationReviewDecision) || ((size(policyViolations) > 0 ) &amp;&amp; (policyScheme.equals("fail")))) </Source> </Script> </Transition>`
rule	Evaluates the return value of a workflow rule to determine step transition. Examples: `<Transition to="Process Approval" when="rule:RequireApprovalRule">`
call	Evaluates return value of a call to a workflow library method to determine step transition. Example: `<Transition to:"Check Status" when="call:requiresStatusCheck" />`
ref	Evaluates a defined, Boolean, workflow variable to determine step transition. Example: `<Transition to="Refresh Identity" when="ref:doRefresh"/>`
Unconditional	Specified as last transition option to give a default path for the transition. Example: `<Transition to="Approve"/>`

Step Actions

Most steps involve much more than a name and a transition. Steps also include an action attribute that executes the workflow processing. The action of a step can be a script or can a rule, subprocess, or a call to a workflow library method.

Table 75—Step Actions

Action Type	Description

Script	Similar to scripts in other parts of the workflow XML, the script can be contained within the action attribute or can be nested within the Step in a <Script> block.
	
	Examples:
	
	(1) In-line Script, used only for very short, simple scripts.
	
	`<Step action="script:approvalSet.setAllProvisioned();" icon="Task" name="Post Provision">`
	`   <Transition to="Stop"/>`
	
	`</Step>`
	
	(2) Longer script within nested <script> tag. Used for action scripts of any complexity or length.
	
	`<Step name="Start" icon="Start" posX="20" posY="20">`
	`   <Script>`
	`      <Source>`
	`         String wfName = wfcontext.getWorkflow().getName();`
	`       System.out.println("Starting workflow: [" + wfName + "]");`
	`      </Source>`
	`   </Script>`
	`   <Transition to="Compile Provisioning Project"/>`
	`</Step>`

Rule	A step can execute a block of Java BeanShell code encapsulated in a reusable workflow Rule. Example: `<Step action="rule:WFRule_verifyIdentity" icon="Task" name="Verify Identity" posX="600" posY="202">`

Table 75—Step Actions

Action Type	Description

Subprocess	When you include a <WorkflowRef> element within the step and reference the SailPoint**.object.Workflow** class and the specific workflow by name, a subprocess is defined.
	
	Example:
	
	`<Step icon="Task" name="Initialize" posX="320" posY="126">`
	`     …`
	`    <WorkflowRef>`
	`        <Reference class="sailpoint.object.Workflow" name="Identity`
	`         Request Initialize"/>`
	`    </WorkflowRef>`
	`    <Transition to="end">`
	`</Step>`

Call	Calls to workflow library methods can be used to do step processing. **Note: Call is the default action option. Therefore the "call:" prefix can be included or omitted.** Example: `<Step action="call:refreshIdentity" icon="Task" name="Refresh Identity" posX="618" posY="242">`

Arguments

Any variables to be passed to a script, rule, subprocess, or library method must be declared as step arguments through <Arg> elements. Similar to other variables, the values for arguments can be specified by string, script, rule, call, or reference. The default specification type is string. Therefore, the “string:” qualifier can be omitted. However, arguments are also commonly passed by referencing workflow variables.

```
Step icon="Task" name="Initialize" posX="320" posY="126">
    <Arg name="w" vaflolue="ref:flow"/>
    <Arg name="formTemplate" value="string:Identity Update"/>
    <Arg name="identityName" value="ref:identityName"/>
     …
    <Description>Call the standard subprocess to initialize the request,
       this includes auditing, building the approvalset, compiling the plan into
       project and checking policy violations.</Description>
    …
    <WorkflowRef>
        <Reference class="sailpoint.object.Workflow" name="Identity Request
           Initialize"/>
    </WorkflowRef>
    <Transition to="end">
```

```
</Step>
```

When an argument is specified as a script, rule, or call, for example, <Arg name="myVar" value="rule:myWFRule"/>, any needed arguments to the script, rule, or library method cannot be explicitly specified.

Because these scripts, rules, and library methods automatically have access to the workflow context object, the scripts can access workflow variables directly through the workflow context get methods. These scripts/rules/methods can also access any step arguments that were defined before them in the step declaration. For example, the method that identifies the value for the Manager argument can use the value in the identityName argument in its processing, if needed. See the following example.

```
<Step icon="Task" name="Processing Step" posX="320" posY="126">
    <Arg name="identityName" value="ref:identityName"/>
    <Arg name="Manager" value="call:getManager"/>
    …
</Step>
```

The following table lists the available Arg attributes

Table 76—Available Arg Attributes

Arg Attribute	Purpose
name	Variable name in process to which the data is being passed.
value	Value to pass into the variable, such as string, script, rule, call, reference.

Return Elements

To return more than one value from a subprocess, you can declare <Return> elements for the step. At a minimum, a return element contains: a **name** attribute and a **to** attribute. The name attribute is the name of the variable in the subprocess workflow and the **to** attribute is the variable name in the calling (current) workflow. If these names are the same in both workflows, a **to** attribute is not required. However, specifying a to attribute is a best practice for clarity.

Use the merge attribute when the variable is a List and the returned values should be appended to the current workflow's list instead of replacing it. Similar to Args, value attribute for return elements can be specified as a string, script, rule, call, or reference. String is the default. If the value argument is omitted, the value of the name variable is copied as-is into the to variable, However, a script/rule/method can be used to transform or modify the value as it is passed.

- **name** attribute — name of the variable in the subprocess workflow
- **to** attribute — variable name in the calling (current) workflow

Note: If these names are the same in both workflows, a to attribute is not required. However, specifying the to attribute is best practice.

```
<Step icon="Task" name="Initialize" posX="320" posY="126">
    <Arg name="flow" value="ref:flow"/>
    <Arg name="formTemplate" value="string:Identity Update"/>
    <Arg name="identityName" value="ref:identityName"/>
     …
    <Return name="project" to="project"/>
    <Return merge="true" name="workItemComments" to="workItemComments"/>
    <WorkflowRef>
        <Reference class="sailpoint.object.Workflow" name="Identity Request
           Initialize"/>
    </WorkflowRef>
    <Transition to="end">
</Step>
```

The following table lists the available Return attributes.

Table 77—Available Return Attributes

Return Attribute	Purpose
name	Variable name in process from which the data is returned.
to	Variable name in the workflow step to which the data is passed.
value	Value to pass into the variable, such as string, script, rule, call, reference.
merge	Flag indicating that the value should be merged into the target variable instead of replacing the variable. This attribute is used for list variables.
local	Only applies to returns on Approvals, see “Approval Steps” on page 200. A flag that indicate the value is passed to local storage within the parent approval and not passed to a workflow case variable. This attribute is used for complex approvals where a work item state is saved for later analysis in a script.

Call

Use calls to workflow library methods to do step processing. Similar to subprocesses, they sometimes require arguments to be passed to them. You declare method arguments the same way as subprocesses. You use Library methods with a call action. See the following example.

```
<Step action="call:refreshIdentity" icon="Task" name="Refresh Identity" posX="618"
posY="242">
    <Arg name="identityName" value="ref:identityName"/>
    <Arg name="correlateEntitlements" value="string:true"/>
   <Description>Add arguments as necessary to enable refresh features. Typically you
      only want this to correlate roles. Don't ask for provisioning since that
      can result in provisioning policies that need to be presented and it's
      too late for that. This is only to get role detection and exception
      entitlements in the cube.</Description>
    <Transition to="Notify"/>
  </Step>
```

The methods available for the call action are those included in the libraries attribute for the workflow element, if specified. If no libraries attribute is specified, the workflow automatically has access to the methods in the Identity, Role, PolicyViolation, and Lifecycle Manager libraries. If other libraries, including custom libraries, are explicitly listed in the libraries attribute, any of the default libraries whose methods are needed by the workflow must also be explicitly included in the list to be available. See “Workflow Library Methods” on page 206 for details about the methods available in each library.

Note: Installations can create custom libraries for commonly used and required business methods. However, custom library methods must be named with unique names that do not conflict with standard library method names. Conflicts resolve as a reference to the standard library method. It is possible to extend a standard library and overload its method names. Extending a standard library is not consider a best practice. Therefore, the best practice is to create new names for nonstandard methods. Creating new names makes it clear that the method is not a standard method.

Wait Attribute

The step wait attribute causes the workflow to pause in its execution for the duration specified. The wait value can be specified as a string, script, rule, call, or reference. String is the default.

```
<Step name="Wait for next check" wait="ref:provisioningCheckStatusInterval">
    <Description>
       Pause and wait for things to happen on the PE side.
       Use the configurable interval to determine how long
       we wait in between checks.
    </Description>
    <Transition to="CheckStatus"/>
  </Step>
```

This attribute creates a special type of step with the sole purpose of creating a pause in the action. Wait steps are commonly used in re-try logic to enable behind-the-scenes processing to occur before the workflow attempts to repeat an action.

Catches Attribute

These steps are not caused through a transition from a previous step. These steps are caused by a thrown message that the steps intercepts or catches. Currently, only a complete message is thrown and can be caught. This process occurs when one of the following items occurs:

- All sequential steps in a workflow are executed to completion.
  OR
- Failure condition results in the termination of the workflow.

```
<Step catches="complete" icon="Task" name="Finalize">
    <Arg name="project" value="ref:project"/>
    <Arg name="approvalSet" value="ref:approvalSet"/>
    <Arg name="trace" value="ref:trace"/>
    <Description>
       Call the standard subprocess that can audit/finalize the request.
    </Description>
    <WorkflowRef>
      <Reference class="sailpoint.object.Workflow" name="Identity Request Finalize"/>
    </WorkflowRef>
    <Transition to="end"/>
```

The primary purpose of these steps is to update the IdentityRequest object, which tracks and reports the status of a LifecycleManager request, making the history of LCM request processing available even after the TaskResult for the workflow was purged.

Each installation can drive custom logic based on catching this complete message.

Approval Steps

Approval is one of the most common actions that a workflow process performs. The IdentityIQ Approval model is constructed to simplify the process of defining an approval structure. Approvals are a special type of step that contain an <Approval> element, specifying how the approval work item is presented for approval.

Some approval steps are designed to get a user's approval on a requested change, as the name implies. However, the approval element can be used any time data needs to be gathered from a user.

Typically, when you use approval steps to gather non-approval data, you use a custom form to:

- Present the work item to the user
  and
- Request the needed information from the user.

For information on creating approval steps, see the section above. Through the XML, the custom form is manually defined within an approval step. You can also specify custom forms for traditional approvals when you need to present the information differently than the standard approval forms layout. See"Workflow Forms" on page 224 for more details on usage of custom forms.

Similar to other Workflow elements, you specify some modifiers as attributes on the approval element and specify other modifiers through nested elements within the approval.

Table 78—Approval Step Attributes

Approval Attribute	Purpose
mode	Specifies how an approval is processed. Mode can be determined from string, script, rule, call, or reference String is the default. The user interface only supports the selection of a string of one of the values listed below.The XML also enables reference to a process variable containing one of those values or the specification of a script, rule, or method call that can determine one of those values programmatically. Valid values are: **serial** - approvers are specified in order and the item is passed to each approver in that order. If any approver in the chain rejects, the item is rejected. **serialPoll** - approvers are specified in order and item is passed to each approver in that order. Data is collected on approvals and rejections. However, if one approver rejects, does not necessarily result in the item being rejected. The action decision is expected to be specified in AfterScript logic. **parallel** - item is sent to all named approvers at one time. The item is rejected if any approver rejects it. **parallelPoll** - item is sent to all named approvers at one time. Data is collected on approvals and rejections but rejection by one does not mean rejection of item. The action decision is expected to be specified in AfterScript logic. **any** - item is sent to all named approvers at one time. The first approver to respond makes the decision for the group.
owner	One or more approvers can be specified by string, script, rule, call, or reference. STring is the default. The mode determines how and when the item is submitted to each listed owner when more than one is specified.
renderer	JSF include to render the work item details.
return	Comma-separated values (CSV) list of variable names to copy from completed work items back into workflow.
send	CSV list of variable names to include in the work items.
description	Defines work item description. For nested approvals, child approvals use the work item defined by the parent approval unless the child approval defines its own work item. You can set the description by string, script, rule, call, or reference String is the default.
validation	Used to validate any information the user entered during the approval. This attribute can be specified as string, script, rule, call, or reference. Script is the default. You generally use a nested validationScript element instead of a validation argument.
Nested Tag within Approval Element	

Table 78—Approval Step Attributes

Approval Attribute	Purpose
AfterScript	Provides instructions for additional processing to be done on the item after the approval is complete, and only if approved. Often uses methods in the Approval Rule Library and LCM Workflow Rule Library. If those methods are to be used, the rule libraries must be explicitly included in the workflow using the <RuleLibraries> element. **Note: ParallelPoll and serialPoll items always execute this script after all responses are collected. With either of these modes, the logic in this script should aggregate the results and determine if the item should be approved or rejected. The business determines the criteria for approval or rejection, for example majority rule, any approval=approval, etc.** In either poll mode, the AfterScript is inherited by child approvals if one is not specified.In other modes, child approvals do not inherit the after script.
InterceptorScript	This script is more complex than the AfterScript and is used less often. The script is called in several places in the approval processing: at the approval start, pre-Assimilation, post-Assimilation, when the work item is archived, and at the end of the approval. The stage of the processing is passed to the script as an argument called method that can be used to determine what the script should do at that time. The workflow context's args are also passed to the script. Method values for conditional analysis within InterceptorScript logic: startApproval preAssimilation postAssimilation archive endApproval**If an InterceptorScript and AfterScript exist, the InterceptorScript postAssimilation logic launches before the AfterScript.**
validationScript	Script to perform validation on the work item. For example, you can use this script to validate any data the user enters on the approval before the data is assimilated. This script is inherited by any child approvals.
Source	Nested within the AfterScript, InterceptorScript, and validationScript tags and contains the java BeanShell source for the script.

Table 78—Approval Step Attributes

Approval Attribute	Purpose
Arg	Arguments available to the approval action. Specified by string, script, rule, call, or reference. Most variables are passed to approval through send list. However, args that require any transformation must be sent through an Arg element. Additionally, the following args defined with reserved system names are passed through the Arg element with that name specified: workItemRequester workItemDescription workItemType workItemTargetId workItemTargetName workItemTargetClass workItemDisableNotification workItemNotificationTemplate workItemEscalationTemplate workItemReminderTemplate workItemEscalationRule workItemEscalationStyle workItemHoursTillEscalation workItemHoursBetweenReminder workItemMaxReminders workItemPriority workItemIdentityRequestId workItemArchive
Return	Return value defines how things should be assimilated from a work item back into the workflow case. This attribute is an alternative to the return attribute CSV of variables. It is more complex and also more powerful. This attribute is rarely used in approvals. It is most often used when returning an approval work item variable to a workflow variable of a different name or when you need to transform the variable contents of a work item with a script. The use of these types of return elements follows the same rules as step returns from steps that subprocesses, with addition of local attribute options. See, “Return Elements” on page 197.

The following basic approval step example presents an account change to the identity's manager for approval. The AfterScript records the approval decision and creates an audit record.

```
<RuleLibraries>
    <Reference class="sailpoint.object.Rule"  name="Approval Library"/>
    <Reference class="sailpoint.object.Rule"  name="LCM Workflow Library"/>
</RuleLibraries>

<Step icon="Approval" name="Manager Approval">
<Approval mode="serial" owner="script:getManagerName(identityName, launcher,
fallbackApprover);" renderer="lcmWorkItemRenderer.xhtml"
send="approvalSet,identityDisplayName,identityName,policyViolations">

<Arg name="workItemDescription" value="Manager Approval - Account Changes for User:
$(identityDisplayName)"/>
<Arg name="workItemNotificationTemplate" value="ref:managerEmailTemplate"/>
<Arg name="workItemRequester" value="$(launcher)"/>

<AfterScript>
        <Source>

          import sailpoint.workflow.IdentityRequestLibrary;
          assimilateWorkItemApprovalSet(wfcontext, item, approvalSet);
IdentityRequestLibrary.assimilateWorkItemApprovalSetToIdentityRequest(wfcontext,
approvalSet);
auditDecisions(item);

</Source>
      </AfterScript>

</Approval>

    <Description>
      If approvalScheme contains manager, send an approval for all
      requested items in the request. This approval will get the entire
      approvalSet as part of the workitem.
    </Description>

<Transition to="Build Owner ApprovalSet"
    when="script:isApprovalEnabled(approvalScheme, &quot;owner&quot;)"/>
<Transition to="Build Security Officer ApprovalSet"
    when="script:isApprovalEnabled(approvalScheme, &quot;securityOfficer&quot;)"/>
```

```
<Transition to="end"/>


  </Step>
```

Note: In the AfterScript in this example, the methods not qualified by the library name are in the LCM Workflow Rule Library that is available to the workflow through the <RuleLibraries> declaration.

The assimilateWorkItemApprovalSetToIdentityRequest method is part of the IdentityRequestLibrary, this is available to the script through the import of that library in the script.

Library methods called through step action attributes are available through the workflow libraries attribute list,. However, when the library methods are executed from within scripts, the library must be specifically imported for the script.

Nested Approvals

Child approvals created through the user interface are expressed as nested approval elements in the XML.When nested approvals exist, the parent ceases to be an approval of its own.In those case, the sole purpose of the parent approval is to organize and contain the child approvals. The mode on the parent determines how to process the set of peer child approvals.

```
   <Approval mode="string:parallel" name="Approve Region" owner="ref:regionApprover"
        send="identityName,region">
       <Arg name="workItemDescription" value="string:Approve Region for
$(identityName)"/>
       <Approval name="childApproval1" owner="string:Walter.Henderson"
           send="identityName,region"/>
       <Approval name="childApproval2" owner="string:Alan.Bradley"
           send="identityName,region"/>
   </Approval>
```

In the example above, childApproval1 and childApproval2 are processed in parallel. Because both of these child approvals are identical (no custom work item config and no children of their own), the same objective can be accomplished with a single approval with multiple owners:

```
   <Approval mode="string:parallel" name="Approve Region"
       owner="string:Walter.Henderson,Alan.Bradley" send="identityName,region">
       <Arg name="workItemDescription" value="string:Approve Region for
$(identityName)"/>
   </Approval>
```

Nested approvals can be used effectively when different approval levels are implemented with custom configurations and specifications. For example, the workItemConfig for each of the child approvals can be different, which can result in a notification scheme, escalation policy, etc. for the different approvers.

Nested approvals can be governed by a different approval mode from the one used on the master set and/or can contain their own child approval levels. One child approval can be done as an any approval, one that accepts the ruling of the first responder of several listed approvers, while the highest approval level is managed serially. Another child approval can implement custom workItemConfigs for its own child approvals. The example below illustrates all of these concepts.

Nested approvals can be used effectively when different approval levels are implemented with custom configurations and specifications. For example, the workItemConfig for each of the child approvals can be. The following example that illustrates all of these concepts.

```
   <!-- Approval submitted to HR and to supervisor and manager in serial manner -->
   <Approval mode="string:serial" name="Approve Region" owner="spadmin"
       send="identityName,region">
      <Arg name="workItemDescription" value="string:Approve Region for
$(identityName)"/>

      <!-- HR Personnel approve region (whoever responds first makes decision) -->
      <Approval name="HRApproval" mode="string:any"
           owner="ref:HRApprovers" send="identityName,region"/>

      <!-- Supervisor and Manager approve region serially after HR approves -->
     <!-- Each has a different email template (work item config) for notification -->
    <Approval mode="string:serial" name="SupMgrApproval" send="identityName,region">
         <Approval name="Supervisor" send="identityName,region" owner="Tom.Jones">
           <WorkItemConfig escalationStyle="none">
              <NotificationEmailTemplateRef>
                 <Reference class="sailpoint.object.EmailTemplate"
                   name="SupervisorApprovalEmail"/>
              </NotificationEmailTemplateRef>
           </WorkItemConfig>
         </Approval>
         <Approval name="Manager" send="identityName,region" owner="Mary.Peterson">
           <WorkItemConfig escalationStyle="none">
              <NotificationEmailTemplateRef>
                 <Reference class="sailpoint.object.EmailTemplate"
name="ManagerApprovalEmail"/>
              </NotificationEmailTemplateRef>
           </WorkItemConfig>
         </Approval>
      </Approval>
   </Approval>
```

This ability to nest approvals, with options to assign different approval modes and work item configurations to each, enables implementers to create highly customized approval structures to meet the needs of the installation.

Workflow Library Methods

Workflow Libraries are sets of compiled java methods. To be accessible to workflows, these libraries must be specified as a comma separated list in the libraries attribute of the workflow element. The classes for libraries are named as follows: SailPoint**.workflow.[library]Library.class**. Only the [library] portion is specified in the libraries attribute.

The following example makes methods from the SailPoint**.workflow.IdentityLibrary.class** accessible to the workflow.

```
Example:

<Workflow libraries="Identity" explicitTransitions="true" name="Hello World
Workflow" type="IdentityUpdate">
```

Note: If no Libraries attribute is specified on the Workflow element, the workflow can access the Identity, Role, PolicyViolation, and LCM libraries by default.

The following table lists the workflow libraries and the methods available. Although the Standard Workflow Handler is not technically a library, the methods in it are accessible to every workflow and are called through the same syntax as library methods.

Standard Workflow Handler

Table 79—Standard Workflow Methods

Method / Usage	Description	Expected Args *=required
Object getProperty(WorkflowContext wfc)	Returns value of the named system property.	name*
public Object isProperty(WorkflowContext wfc)	Returns true if the named system property has a value.	name*
public Object getMessage(WorkflowContext wfc)	Returns localized message for use in task results	message* type (severity) arg1-arg4 (up to 4 parameters for the message)
public Object addMessage(WorkflowContext wfc)	Adds message to the workflow case.	message* type (optional severity) arg1-arg4 (up to 4 parameters for the message)
public Object addLaunchMessage(WorkflowContext wfc)	Adds message to workflow case that is displayed in the user interface. Not kept in task result. For example, **Request was submitted**.	message* type (optional severity) arg1-arg4 (up to 4 parameters for the message)
public Object setLaunchMessage(WorkflowContext wfc)	Replaces previously added launch message with a new message based on new state.	message* type (optional severity) arg1-arg4 (up to 4 parameters for the message)
public Object log(WorkflowContext wfc)	Sends something to log4j.	message* level*
public Object print(WorkflowContext wfc)	Prints text to the console.	message*

Table 79—Standard Workflow Methods

Method / Usage	Description	Expected Args *=required
public Object audit(WorkflowContext wfc)	Creates an audit event. Enables workflows to put custom entries in audit log, which displays in the user interface.	source* action* target string1 - string4
public Object sendEmail(WorkflowContext wfc)	Sends an email message.	to* cc bcc from subject body template* templateVariables sendImmediate exceptionOnFailure
public Object launchTask(WorkflowContext wfc)	Launches a defined task.	taskDefinition* taskResult sync (true=synchronous execution)
public Object scheduleRequest(WorkflowContext wfc)	Launches a generic event request.	requestDefinition* requestName (name to assign to request) scheduleDate scheduleDelaySeconds owner

Table 79—Standard Workflow Methods

Method / Usage	Description	Expected Args *=required
public Object scheduleWorkflowEvent(WorkflowContext wfc)	Launches a workflow event request.	requestName (name to assign to request) scheduleDate scheduleDelaySeconds owner workflow* (name of workflow to launch) caseName (optional case name to override default)
public Object commit(WorkflowContext wfc)	Commits a transaction. Not commonly needed in workflows. Most commonly used for role approvals.	creator archive
public Object rollback(WorkflowContext wfc)	Rolls back a transaction. Not commonly needed in workflows. Most commonly used for role approvals.	none

Identity Library

Table 80—Identity Library Methods

Method / Usage	Description	Expected Args *=required
public String getManager(WorkflowContext wfc)	Returns the name of the manager for the specified identity.	identityName
public Object calculateIdentityDifference(WorkflowContext wfc)	Derive a simplified representation of the changes made to an identity for an approval work item.	oldRoles newRoles plan approvalSet

Table 80—Identity Library Methods

Method / Usage	Description	Expected Args *=required
private void addLinksInformation(WorkflowContext wfc)	Modifies workflow context lists of links (accounts) to be added, moved, or removed for the identity as a result of the provisioning plan.	linksToAdd linksToMove linksToRemove plan
public List<Map<String,Object>> checkPolicyViolations(WorkflowContext wfc) Evaluate policy violations that can be incurred by the provisioning plan/project's actions	Evaluates policy violations that the provisioning plan/project actions can incur.	policies identityName* project plan (either plan or project is required)
public void activateRoleAssignment(WorkflowContext wfc)	Assigns a role or roles to the identity.	identity* (ID) role* (ID) detected (Boolean indicating if role was detected vs. assigned)
public void deactivateRoleAssignment(WorkflowContext wfc)	Removes role assignments from the identity.	identity* (ID) role* (ID) detected (Boolean indicating if role was detected vs. assigned)
public void refreshIdentity(WorkflowContext wfc)	Performs an identity refresh on one identity.	identity (ID) identityName (either identity or identityName is required)
public void refreshIdentities(WorkflowContext wfc)	Performs an identity refresh on a set of identities. Can specify one or more identityNames, a filterString, or a list of roles. Processes the first of the above listed options that is non-null.	identityName identityNames (CSV) filterString identitiesWithRoles (CSV) (any one of these 4 is required)
public Object compileProvisioningProject(WorkflowContext wfc)	Compiles a provisioning plan into provisioning project.	plan identityName

Table 80—Identity Library Methods

Method / Usage	Description	Expected Args *=required
public Object buildProvisioningForm(WorkflowContext wfc)	Creates a form to display provisioning policy questions. When requiredOwner is passed as an argument, a form owned by this user is returned. If no more forms for this user exist, null is returned. When preferredOwner is passed as an argument, a form owned by this user is returned. If there are no remaining forms for that owner, a form owned by a different user can be returned.	project* template (name of form to serve as page template) owner preferredOwner (owner or preferredOwner required but mutually exclusive)
public Object assimilateProvisioningForm(WorkflowContext wfc)	Collects data from completed a provisioning form and stores answers with questions on provisioningProject.	project* form*
public Object assimilateAccountIdChanges(WorkflowContext wfc)	Updates ApprovalSet with any changes to accountIDs.	project* approvalSet
public Object buildPlanApprovalForm(WorkflowContext wfc)	Builds a form that represents all attributes in a provisioningPlan for an approval before the provisioning occurs.	plan* template
public Object assimilatePlanApprovalForm(WorkflowContext wfc)	Collects data from a form and puts the data back into the provisioningPlan. Assumes buildPlanApprovalForm.	form plan*
public Object provisionProject(WorkflowContext wfc)	Called by the Identity Update and LCM Workflows after provisioning forms are completed. Provisions the remaining items in the project.	project* noTriggers (Boolean)

Table 80—Identity Library Methods

Method / Usage	Description	Expected Args *=required
public Object finishRefresh(WorkflowContext wfc)	Called by the Identity Refresh workflow, after approvals are done and account completion attributes are gathered. Provisions what it can and completes the refresh process.	identitizer refreshOptions (map of args for creating new Identitizer if needed) previousVersion project
public Object buildApprovalSet(WorkflowContext wfc)	Called by the Lifecycle Manager workflows. Builds a simplified ApprovalSet representation of the items in the provisioning plan.	plan*
public Object processApprovalDecisions(WorkflowContext wfc)	Processes decisions made during approval process audit and react. Modifies the project masterPlan and recompiles the project if the recompile argument is set to true.	project* dontUpdatePlan disableAudit approvalSet* recompile
public Object processPlanApprovalDecisions(WorkflowContext wfc)	Processes decisions made during approval process audit and modifies the Used before the plan is compiled into a provisioningProject.	plan* dontUpdatePlan disableAudit approvalSet*
public Object auditLCMStart(WorkflowContext wfc)	Creates an audit event to mark the start of an Lifecycle Manager workflow.	approvalSet* flow (name of applicable UI flow)
public Object auditLCMCompletion(WorkflowContext wfc)	Creates an audit event to mark the completion of anLifecycle Manager workflow.	approvalSet* flow
public void disableAllAccounts(WorkflowContext wfc)	Used by lifecycle events to disable managed accounts for the identity specified in the workflow.	None
public void enableAllAccounts(WorkflowContext wfc)	Used by Lifecycle events to enable all accounts on the identity specified in the workflow.	None

Table 80—Identity Library Methods

Method / Usage	Description	Expected Args *=required
public void deleteAllAccounts(WorkflowContext wfc)	Used by Lifecycle events to delete all accounts on the identity specified in the workflow.	None
public ProvisioningPlan buildEventPlan(WorkflowContext wfc)	Go through all links that the workflow's specified Identity hold and creates a plan to enable or disable all of the Identity's accounts. Specified by **op**.	op* (operation)
public void updatePasswordHistory(WorkflowContext wfc)	Adds a password to the link password history	plan*
public ProvisioningProject assembleRetryProject(WorkflowContext wfc)	Adds any account request for an original provisioning project that are retryable and then adds them to a new provisioning project. Rarely used in custom workflows.	project
public Object retryProvisionProject(WorkflowContext wfc)	Executes the retry provisioning project, created in assembleRetryProject. Rarely used in custom workflow.	project
public Object mergeRetryProjectResults(WorkflowContext wfc)	Merges results from the retry project onto the main project. Called between retries. Rarely used in custom workflow.	project* retryProject*
public Boolean requiresStatusCheck(WorkflowContext wfc)	Identifies if the project contains any Results that are queued with a requestID stored on the result.	project
public Object checkProvisioningStatus(WorkflowContext wfc)	Calls down to the connector for each Result in the plan that is marked queued with a requestID specified.	project

Table 80—Identity Library Methods

Method / Usage	Description	Expected Args *=required
public Integer getProvisioningStatusCheckInterval(Workflow Context wfc)	Compute intervals between status checks for a request. The default is 60 minutes.	none
public Integer getProvisioningMaxStatusChecks(Workflow Context wfc)	Computes the maximum number of status checks permitted during a request. The default is infinite.	none
public Integer getProvisioningMaxRetries(WorkflowContext wfc)	Computes the maximum number of retries permitted during a request. The default is infinite.	none
public Integer getProvisioningRetryThreshold(WorkflowContext wfc)	Computes the retry threshold, the interval between retries, to use for a request. the Default is 60 minutes.	none

The methods are available for use. However these methods are rarely used in a custom workflow. It is recommended that custom workflows the workflow subprocesses instead of calling the library methods directly.

Note: This information is included for reference purposes and to document the purpose of the methods and what is passed to them. These explanations are also included to ensure that customizations do not remove calls to important methods from the subprocess workflows and to ensure that customizations only add other functionality around these method calls.

IdentityRequest Library

Table 81—IdentityRequest Library Methods

Method / Usage	Method / Usage	Expected Args *=required
public Object createIdentityRequest(WorkflowContext wfc)	Create s an IdentityRequest object from current workflow context information. Tracks status and history of request processing.	project* flow source policyViolations
public Object updateIdentityRequestState(WorkflowContext wfc)	Modifies the IdentityRequest's state.	identityRequestId

Table 81—IdentityRequest Library Methods

Method / Usage	Method / Usage	Expected Args *=required
public Object refreshIdentityRequestAfterApproval (WorkflowContext wfc)	Refreshes the IdentityRequest to include the provisioningEngine that processes the item. Updates the state and adds any expanded attributes that are provisioned.	project
public Object refreshIdentityRequestAfterProvisioning (WorkflowContext wfc)	After provisioning, copies interesting task result information back to the IdentityRequest.	project
public Object refreshIdentityRequestAfterRetry (WorkflowContext wfc)	Scans the retry project and updates the IdentityRequestItem retry count.	project
public Object completeIdentityRequest (WorkflowContext wfc)	Marks IdentityRequest status complete. Puts final plan in request and refreshes the request based on the final project.	project policyViolations autoVerify (Boolean)

Approval Library

Table 82—Approval Library Methods

Method / Usage	Method / Usage	Expected Args *=required
public SailPointObject getObject(WorkflowContext wfc)	Returns the object being approved.	none
public String getObjectClass(WorkflowContext wfc)	Returns the simple class name of the object being approved.	none
public String getObjectName(WorkflowContext wfc)	Returns the name of the object being approved.	none
public SailPointObject getCurrentObject(WorkflowContext wfc)	Returns the current persistent version of the object held in the workflowCase (approvalObject).	none
public Identity getObjectOwner(WorkflowContext wfc)	Returns the current owner of the object being approved. Uses database lookup.	none

Table 82—Approval Library Methods

Method / Usage	Method / Usage	Expected Args *=required
public Identity getNewObjectOwner(WorkflowContext wfc)	Returns the object owner. In the workflow context, the owner could be different than the database-recorded owner.	none
public String getObjectOwnerName(WorkflowContext wfc)	Returns name of ObjectOwner from getObjectOwner.	none
public String getNewObjectOwnerName(WorkflowContext wfc)	Returns name of NewObjectOwner from getNewObjectOwner.	none
public boolean isOwnerChange(WorkflowContext wfc)	Return true if object being approved has had an owner change.	none
public boolean isSelfApproval(WorkflowContext wfc)	Returns **True** if the user who launches workflow is the same as the owner of the object being approved. Used to bypass an owner approval. Assumes that the user will approve if the user is the one who is initiating the request.	none

Policy Violation Library

Table 83—Policy Violation Library Methods

Method / Usage	Method / Usage	Expected Args *=required
public Object delete(WorkflowContext wfc)	Deletes the current approval object associated with this workflow.	none
public Object ignore(WorkflowContext wfc)	Ends the workflow associated with the current approval object without performing any actions.	none
public Object mitigateViolation(WorkflowContext wfc)	Mitigates by temporarily allowing a policy violation.	expiration* comments
public Object getRemediatables(WorkflowContext wfc)		none

Table 83—Policy Violation Library Methods

Method / Usage	Method / Usage	Expected Args *=required
public Object getRemediatables(WorkflowContext wfc)		remediator actor Use if remediator argument is not specified and actor is. Use remediator in new method calls. comments remediations*

Role Library

Table 84—Role Library Methods

Method / Usage	Method / Usage	Expected Args *=required
public Object launchImpactAnalysis(WorkflowContext wfc)	Starts an impact analysis task for a role in workflow.	none
public Object getRoleDifferences(WorkflowContext wfc)	Calculates the differences between a role held in workflow and the database version of the role.	none
public Object auditRoleDifferences(WorkflowContext wfc)	Creates one audit event for each attribute difference between role states. Compares workflow vs database.	source action target string1
public Approval buildOwnerApproval(WorkflowContext wfc)	Sets up an approval for the owner of an object. Currently used only for roles.	none

Table 84—Role Library Methods

Method / Usage	Method / Usage	Expected Args *=required
public List<Approval> buildApplicationApprovals(WorkflowContext wfc)	For role approvals only. Builds an approval structure for the owners of each application referenced in the role profiles. Normally processed as parallelPoll to allow application owners to submit comments or modify the role without terminating the approval process.	none
public void enableRole(WorkflowContext wfc)	Marks role as enabled.	role (name)
public void disableRole(WorkflowContext wfc)	Marks role as disabled.	role (name)

public void setRoleDisabledStatus(WorkflowContext wfc)	Marks role with disabled status indicated in the disabled arg.
	true = disabled
	false = enabled
		role (name)
		
		disable (Boolean)

public void removeOrphanedRoleRequests(WorkflowContext wfc)	Removes incomplete requests. Used to activate/deactivate roles that no longer exist.	none
public String getApprovalAuditAction(WorkflowContext wfc)	Called by the post-approval audit steps, Audit Failure and Audit Success, of Role Modeler. Owner Approval workflow to determine what type of action should be recorded in audit log. If the role is marked as disabled, returns disableRole. if the role is NOT marked as disabled, returns updateRole .	none

LCM Library

Currently, the Lifecycle Manager Library contains no public methods. All of its methods were moved to the Standard Workflow Handler.

Monitoring Workflows

After a workflow is initiated, the workflow can launch to completion quickly. Sometimes a workflow can take additional time to complete its specified actions. Approval steps often create a delay in the processing while the workflow waits for the approver to review the work item and make a decision on it.

To observe a workflow in flight and understand how much of the process is complete and what actions are pending, You can examine the Task Result for the workflow on the **Monitor -> Tasks -> Task Results** page. The TaskResult for a workflow exists for a period of time following the successful completion of the workflow. Based on the retention period set, the TaskResult can be purged soon after the process launches to completion. While the workflow is still in progress, the TaskResult continues to exist and can be examined for current step and status information.

Viewing the Workflow Case XML

You can examine the workflow case in XML format from the IdentityIQ console or from the Debug pages.The status of each step can then be determined from the data recorded in the workflow case.

To get the **workflowcase** XML from the console:

1. Launch the console.
2. List the workflow cases.
3. Get the specific workflow case in question by name. See the following example.

```
iiq console

> List workflowcase

[system will list all in-flight workflowcases by ID and name]

> get workflowcase "[workflowcase name]"
[system will display the XML for the workflow case]
```

To view the workflowcase XML from the IdentityIQ Debug pages:

1. Select WorkflowCase from the object list.
2. Click the specific workflow case from the list to display its XML.

Chapter 15: Advanced Workflow Topics

This chapter includes the following advanced Workflow topics:

- Loops within Workflows
- Launching a Workflow from a Task
- Workflow Forms

Loops within Workflows

A loop occurs when a step transitions back to a step that executed previously. The state of that step is re-initialized and the step is executed again. A loop can transition back any number of steps. You define a loop transition the same way you would any other transition. However, you must just select a target step that appears before the loop transition in the process designer.

In most case, when you create a loop transition, you want to give it a conditional <i>When</i> expression. If a loop transition is unconditional, the workflow can enter an infinite loop and not be able to finish.

Launching Workflows from a Task or Workflow

You can launch workflows from tasks or other workflows without using a system event to trigger the workflow.

Workflows Launched from Custom Tasks

You can launch workflows from a custom task in IdentityIQ. Because tasks are compiled java classes, the custom task must be written as a Java method.

To create a workflow from a custom task:

1. Create a WorkflowLaunch object in the Java method.
2. Populate the object with the data the workflow requires.
3. Use the <i>Workflower</i> class to launch the workflow.

It is often necessary for one workflow to launch another workflow. This can be performed in Beanshell using code similar to the previous example. However, using the workflow library method <i>scheduleWorkflowEvent</i> is easier. Not only does this method launch a workflow, it also allows you to delay the launch until a time in the future.

To have one workflow to launch another workflow, create a step and select scheduleWorkflowEvent as the action. This method requires the following arguments:

```
import java.util.HashMap;
import sailpoint.api.sailpointContext;
import sailpoint.api.Workflower;
import sailpoint.integration.ProvisioningPlan;
```

```
                  import sailpoint.integration.ProvisioningPlan.AccountRequest;
                  import sailpoint.integration.ProvisioningPlan.AttributeRequest;
                  import sailpoint.object.Identity;
                  import sailpoint.object.Workflow;
                  import sailpoint.object.WorkflowLaunch;
                  import sailpoint.tools.GeneralException;
                  import sailpoint.tools.xml.XMLObjectFactory;


                  HashMap launchArgsMap = new HashMap();

                  String myIdentityName = "T339222";
                 Identity myIdentity = context.getObjectByName(Identity.class,
  myIdentityName);

                  //Create Provisioning Plan and add needed attribute values
                  ProvisioningPlan plan = new ProvisioningPlan();
                  plan.setIdentity(myIdentity);
                  AccountRequest accountRequest = new AccountRequest();
                  AttributeRequest attributeRequest = new AttributeRequest();

                  accountRequest.setApplication("IIQ");
                  accountRequest.setNativeIdentity(wbIdentity);
                  accountRequest.setOperation("Modify");

                  attributeRequest.setOperation("Add");
                  attributeRequest.setName("assignedRoles");
                  attributeRequest.setValue("Benefits Clerk");

                  accountRequest.add(attributeRequest);
                  plan.add(accountRequest);

                  //Add needed Workflow Launch Variables to map of name/value pairs
                  launchArgsMap.put("allowRequestsWithViolations","true");
                  launchArgsMap.put("approvalMode","parallelPoll");
                  launchArgsMap.put("approvalScheme","worldbank");
                  launchArgsMap.put("approvalSet","");
                  launchArgsMap.put("doRefresh","");
                  launchArgsMap.put("enableRetryRequest","false");
                  launchArgsMap.put("fallbackApprover","admin");
                  launchArgsMap.put("flow","RolesRequest");
                  launchArgsMap.put("foregroundProvisioning","true");
                  launchArgsMap.put("identityDisplayName","John.Smith");
                  launchArgsMap.put("identityName","John.Smith");
                  launchArgsMap.put("identityRequestId","");
                  launchArgsMap.put("launcher","admin");
                  launchArgsMap.put("notificationScheme","user,requester");
                  launchArgsMap.put("optimisticProvisioning","false");
                  launchArgsMap.put("plan",plan);
                  launchArgsMap.put("policiesToCheck","");
                  launchArgsMap.put("policyScheme","continue");
                  launchArgsMap.put("policyViolations","");
                  launchArgsMap.put("project","");
                  launchArgsMap.put("requireViolationReviewComments","true");
                  launchArgsMap.put("securityOfficerName","");
                  launchArgsMap.put("sessionOwner","admin");
                  launchArgsMap.put("source","LCM");
```

```
               launchArgsMap.put("trace","true");
               launchArgsMap.put("violationReviewDecision","");
               launchArgsMap.put("workItemComments","");

               sailpoint.object.ProvisioningPlan spPlan = new
sailpoint.object.ProvisioningPlan();
               spPlan.fromMap(plan.toMap());
               launchArgsMap.put("plan", spPlan);

               //Create WorkflowLaunch and set values
               WorkflowLaunch wflaunch = new WorkflowLaunch();
               Workflow wf = (Workflow)
context.getObjectByName(Workflow.class,"myWorkflowName");
               wflaunch.setWorkflowName(wf.getName());
               wflaunch.setWorkflowRef(wf.getName());
               wflaunch.setCaseName("LCM Provisioning");
               wflaunch.setVariables(launchArgsMap);


               //Create Workflower and launch workflow from WorkflowLaunch
               Workflower workflower = new Workflower(context);
               WorkflowLaunch launch = workflower.launch(wflaunch);

               // print workflowcase ID (example only; might not want to do this in
the task)
               String workFlowId = launch.getWorkflowCase().getId();
               System.out.println("workFlowId: "+workFlowId);
```

Workflows Launched by Other Workflows

Installations often have one workflow start another workflow using the scheduleWorkflowEvent method in the Standard Workflow Handler. One of the initiating workflow steps launches the method through a call action.

Arguments to the step including the following:

Table 85— scheduleWorkflowEvent Arguments

Name	Value
workflow	Name of the workflow to launch.
requestName	Name to be assigned to the request. **Note: If not specified, the name of workflow is the default.**
scheduleDate	Date and time you want the workflow to launch. Must be specified with a java.util.Date value. If this argument is not set, the workflow launches immediately.
scheduleDelaySeconds	An alternative to using scheduleDate. When set, the value is the number of seconds to delay before launching the workflow.
caseName	Specify a user friendly name for workflowCase to be displayed in the user interface. **Note: If no name is specified, the default is the name of workflow.**

Table 85— scheduleWorkflowEvent Arguments

Name	Value
launcher	Name of the identity to be displayed as the <i>launcher</i> of the new workflow case. If this argument is not specified, the launcher of the initiating workflow is used.

A workflow that is launched by another workflow is different from a workflow that is launched as a subprocess. If a workflow is launched as a subprocess, the calling workflow waits until the subprocess is completed. After the workflow returns control to the caller, the processing continues.

A workflow that is launched by another workflow causes a completely separate workflow to begin launching. After the new workflow is started, the original or calling workflow moves on to its next step.

Workflow Forms

Standard work item forms are available for presenting approval or other data requests to approvers. However, some installations prefer to use custom forms for these activities. Based on the type of the data collection effort, a custom form might be required. You can build a custom form using a <Form> element in the XML that is embedded within the <Approval> element.

Note: The <Approval> element can be used to collect data from a user, even if the workflow is not an approval. You generally use custom forms for these activities because the normal approval forms do not apply. However, you can also use custom forms for traditional approval activities when you need a different presentation format.

The basic elements in a Form definition are:

```
<Form>
   <Attributes>(map of name/value pairs that influence the form renderer)
   <Button> (determine form processing actions)
   <Section>(Subdivision of form; can contain nested Sections and Fields)
    <Field>(can contain Attributes map, Script to set value, Allowed Values Definition
script, and Validation Script)
```

For detailed information about working with forms, see “Forms” on page 23.

Process Variable and Step Forms

You use forms added to steps on the **Process Designer** tab in the Business Process Editor to request required data from a user that a process needs. For example, you can add a form to a step to request a value for a missing attribute.

However, to present information on the Basic Views of the **Process Variables** tab and the **Arguments** tab of the Step Editor, you use process variable and step forms.

To simplify the information displayed on the Process Variables tab:

- Variables are displayed in more logical groups.
- Variables that are rarely, if ever, modified are hidden.

Changes made in the Basic View are persisted to the Advanced View and more complex configuration can be performed there if needed.

The step forms are referenced from the workflows or stepLibraries. These forms define the form that is presented on the **Arguments** tab of the Step Editor panel and works similar to the process variable forms.

Both of these forms are referenced from workflows using the configForm variable. The forms can be defined, viewed and edited on the IdentityIQ debug page.

Managing Reports

This section contains the following information:

Chapter 16: Reports Introduction

Reports provide an at-a-glance view of the data in the IdentityIQ instance, which helps the organization manage system access and the compliance process. Out of the box, IdentityIQ includes a set of core reports in template form. Individual users and organizations can specify and save customized instances of these templates and run these reports on a scheduled or ad-hoc basis. Additionally, custom reports can be created to meet the needs of each customer.

IdentityIQ includes a reporting architecture that simplifies the process of creating custom reports. Basic reports can be created quickly through an XML specification. A variety of hooks are available for introducing more complex logic where it is needed to produce the desired report output. The standard report templates that are part of the product are modeled with this same XML specification structure and can serve as helpful examples of how custom reports should be structured.

This document explores the report interface and the process of specifying filters to create customized instances of the available report templates. It also describes the procedures required to create custom report templates that can also be used to create customized report instances with specific saved filters.

Report Terminology

This set of terms helps clarify the discussion of the options available for creating custom reports and customizing report templates by eliminating confusion around which option is which. The following terms are used:

- **Report Templates**: Out-of-the-box reports as provided with the standard IdentityIQ product. These report templates are on the Reports tab of the Reports window. The reports can be run directly or edited to create customized versions. These reports are also referred to as out-of-the-box reports, standard reports, or standard report templates.
- **Custom Reports**: Customer-specific reports developed by or specifically for a single customer through a custom Task Definition specification. These reports are on the Reports tab of the Reports window after they are saved in IdentityIQ. These report are also referred to as custom report templates.
- **Customized Report Instances** — User-specific report versions with pre-specified parameters. These reports are on the My Reports tab of the Reports window. Instances apply to out-of-the-box reports and custom reports. The terms customized report or instance can also be used in discussing these report specifications.

Chapter 17: Report Use and Customization

IdentityIQ includes a number of standard reports that are helpful in monitoring and managing compliance and provisioning activities. These reports can be run with or without filter specification. For example, the Uncorrelated Accounts Report can run with no filters and return the list of uncorrelated accounts for all applications in the system, or the report user can specify filters on the report to restrict the results to a subset of applications. The unfiltered, standard version of each report is accessible and able to be run from the Report window's Reports tab. When a user chooses to add filters, that report configuration is saved as a customized report instance on the My Reports tab.

To access the Reports page, from the Navigation menu bar, go to **Intelligence -> Reports**.

Reports Tab

The first tab visible on the Reports window is the My Reports page. However, the first time a user access the Reports page, the My Reports list is empty because My Reports only shows the customized reports you created and saved based on a report template. The second tab, the Reports tab, is where a new user must start interacting with reports.

The Reports tab lists all of the available report templates, grouped by report category. The out-of-the-box report categories are:

- Access Review and Certification Reports
- Account Group Reports
- Activity Reports
- Administration Reports
- Configured Resources Reports
- Identity and User Reports
- Policy Enforcement Reports
- Risk Reports
- Role Management Reports

For each report template, the report name and a brief description of its contents are shown. Reports can be run directly from this page or can be scheduled to run at some point in the future, either once or on a repeating, scheduled basis. Any report initiated (immediately or scheduled) from this page is run with no filters applied. In other words, the report runs for all system objects to which that report applies (all roles, all Identities, all policies, all access reviews, etc.).

To run a report with no filters, right click the report on this window and then choose **Execute** to run it once immediately or Schedule to set it up to run in the future or on a repeating basis.To completely remove the report from the system, click **Delete**.

Alternatively, filters and other specifications can be applied to a report and it can be saved as a new report instance with those parameters already in place. Click the report name in the list or right-click the report and choose **Save as New Report**. Both of these options open the Edit Report page that, displays the available filters and parameters for the report.

Edit Report Page

The Edit Report page allows the user to specify filters for the report, saving that configuration as a customized report instance for future re-use. In the new reporting architecture, every report specification is separated into multiple Sections. Every report specification includes a Standard Properties and Report Layout section as the first and last sections, respectively. Any parameters specific to a given report are specified in one or more sections between these two. Navigate between the “section” pages by clicking the desired page in the Sections list or by clicking the Next and Previous buttons at the bottom of the Edit Report page.

Standard Properties

This common set of properties applies to every report, so the Standard Properties page is presented as part of the Edit Report page for every report.

The table below lists the fields on the Standard Properties page and describes their usage.

Table 86—Reports Standard Properties

Field Name	Description	Required?
Name	The name for the report instance - shown on the **My Reports** window as the report's name and on the Report Results page when the report is run	Yes
Previous Result Action	Determines what is done with the results of previous runs of this report when it is run again **Rename Old** — renames existing report results by appending a numeric value to them **Rename New** — renames the run results of the new report by appending a numeric value to it **Delete** — deletes any previously generated report results for this report, replacing them with the new results **Cancel** — prevents the report from running if a previous result exists	Yes
Description	Brief description for the report - shown on the **My Reports** window as the report's description (defaults to the report template's description)	No
Allow Concurrency	Determines whether more than one instance of the report is allowed to be running at a time	No
Scope	Assigns a scope to the report which allows it to be seen and used by any user authorized to that scope; this is the easiest way to implement report sharing	No
Email Recipients	Names Identities to whom the report results should be emailed when it is generated; these users can view the report results but are not required by the system to act on them in any way	No

Table 86—Reports Standard Properties

Field Name	Description	Required?
Require Signoff	Determines whether anyone is required to sign off on the report; when this is selected, the **Signoff Properties** section of the form appears where the email template and signers are specified; a work item is created for each of these signers	No
Initial Notification Email	Specifies the email template for the email that is sent to notify the signers of the report work item created for them	Yes, if signoff required
Escalation	Specifies whether reminders should be sent or escalation should be enacted for the signoff work items. If either or both are selected, additional parameters specifying the timing of reminders/escalation are displayed and must be entered	Yes (though None is the default)
Signers	Specifies the Identities who must sign off on this report.These Identities receive a notification email and a work item is created.	Yes, if signoff required

Report Layout

The **Report Layout** page shows the columns that are available for inclusion on the report and the columns selected for inclusion on the report. It allows the user to select a sort field and a group field if desired. It also allows suppression of either the summary or the detail information as needed.

These are the fields on the Report Layout page, along with descriptions of their usage.

Table 87—Report Layout Fields

Property Name	Description	Required?
Sort By	Specifies the field by which the detail records in the report should be sorted; if a Group By field is also selected, grouping supersedes sorting and the records within each group are sorted by the Sort By field; only columns marked as sortable (sortable=”true” in XML) are included in this selection list	No
Group By	Specifies the field by which the detail records in the report should be grouped; the group field value is displayed as a section header within the report body in the on-screen display of the report and in the PDF (not in the downloaded CSV format) ; only columns marked as sortable (sortable=”true” in XML) are included in this selection list	No
Columns	Lists columns available for or included in the report; this section is split into two lists: the left list indicates available columns that are not selected for inclusion in the report body. The right list displays the columns that will be in the report in the order in which they will appear. Column names can be dragged from one side to the other and can be reordered within the list with drag-and-drop. The arrow buttons between the two lists can also be used to move columns around.	Yes
Disable Report Summary Display	Suppresses the summary section of the report output	No

Table 87—Report Layout Fields

Property Name	Description	Required?
Disable Report Details Display	Suppresses the detail section of the report output	No

Report-Specific Parameters

Most reports implement at least one page between the Standard Properties and Report Layout pages. These are named differently for each report, and some reports implement several pages while others include only one. These pages allow the user to specify filter parameters for the report instance. For example, the Uncorrelated Accounts Report contains one report-specific settings page called **Uncorrelated Accounts Parameters**; this page enables the user to specify the **Application** for which they want to see the a list of accounts that could not be correlated to existing Identities (from the authoritative application). If no application is selected in this filter, the report shows all uncorrelated accounts from all applications.

In some cases, the **Report Layout** column list will change based on the parameters set on the report-specific parameters pages. For example, the User Account Attributes Report can display account attributes on selected applications. If the application selected has attributes (for example, privileged or service accounts) which other applications don't have, when that application is selected for the report, those columns appear on the Report Layout page for optional inclusion in the report.

Saving and Executing Report Instances

Once a report has been customized with filter parameters, sorting and grouping specifications, and custom column selections or order, it can be saved as a My Reports report instance for future use/re-use. There are several save options available:

- **Save**: saves the report instance specification onto the My Reports tab under the name provided on the Standard Properties window
- **Save and Preview**: saves the report specification onto the My Reports tab and runs a preview of the report, which displays the summary section (unless suppressed) and the first page of detail results (20 records); allows the user to verify that the report shows the type of data they want to see; does not save the report results for later viewing
- **Save and Execute**: saves the report specification onto the My Reports tab and runs the report; the report results are saved to the database and can be recalled from the Report Results tab until they are deleted

Modifying through Preview Mode

The **Save and Preview** option enables the user to modify the report output layout, including rearranging columns, changing the detail sort order, and hiding either the summary or detail section. Any changes made in preview mode can be saved to the report specification, allowing Preview mode to function as an interactive method of reconfiguring the report output. A message at the top of the report prompts the user to choose whether to **Save Changes** or **Cancel Changes** when they alter the appearance of the report preview.

When a report is executed (as opposed to previewed), the final results cannot be reordered in the on-screen display of the report. The report can, however, be downloaded as a CSV and manipulated in a spreadsheet application if desired.

Reports without Preview Option

A few of the reports cannot be viewed in preview mode; this is because the data in these reports cannot be polled without fully executing the report. For example, the Identity Forwarding Report shows the forwarding user for all Identities who have one specified. Because the forwarding property is not searchable, these cannot be counted up front and the report cannot be previewed. In some cases, a report can be previewed unless certain options on it are selected. The Manager Access Review Report, for example, is available for previewing unless the Show Excluded Items option is selected. This option requires a union that cannot be done with the preview option, so previewing must be disabled. In these cases, a message is shown indicating that preview is not available when the user clicks **Save and Preview**.

My Reports Tab

When customized report instances are saved, they appear in the list for the user. From this page, any defined report instance can be opened, edited to change the instance's specifications, and saved with updates (with or without running the instance). Instances can also be used to create new report instances and can be run, scheduled, or deleted through their right-click menu options. The right-click menu options are:

- **Save as New Report** — Creates a new report instance based on an existing instance. Changes made are saved as a new report instance, leaving the existing one intact. This is particularly helpful when a report offers a large number of configuration parameters and a new instance is being created that changes only a few of the options (filters, sorts, etc.).
- **Edit** — Edits the existing report instance in place. This is the same as clicking the report instance in the list
- **Schedule** — Schedules the report instance for future (or repeated) execution.
- **Execute** — Runs the report instance immediately.
- **Delete** — Deletes the instance’s configuration from the system.

Scheduled Reports Tab

When a user chooses to schedule a report template or instance for execution (right-click -> **Schedule**), the New Schedule specification window is displayed. The user must specify a name for the scheduled version of the report and can optionally specify a description. The report execution is scheduled at the date and time entered by the user and can be set to run at various frequencies (once, hourly, daily, weekly, monthly, quarterly, or annually).

Reports that have been scheduled for future or repeated execution appear on the Scheduled Reports tab. Once the scheduled report has executed for the last time (for example, it was originally scheduled to run once in the future and that time arrived so it ran), it is removed from this list; only future-scheduled report executions are shown on this page.

Report Results Tab

The Report Results tab shows the completion status of all reports that have run and are currently available for viewing.

Click any report in the list to view its specific results. From the Report Result window, the report can be viewed on screen, downloaded as a PDF, or downloaded as a CSV file. In the new reporting architecture, the PDF and CSV forms of the report are created during report execution and saved to the database, making the download of either format fast and efficient.

XML Representation of Reports and Instances

Standard report templates are represented in the IdentityIQ object model as TaskDefinition objects. The XML representation of these can be seen through the IdentityIQ Debug pages by selecting Task Definition from the **Object Browser** list and searching for the report's name in the **Name** column.

When a report is saved as a customized instance, a new XML object is created in the system to represent its custom configuration. The instance's XML is far simpler than the template's because it references the template for most of the report generation details.

Details on these XML representations are explored further in the next chapter: *Developing Custom Reports*.

Chapter 18: Developing Custom Reports

IdentityIQ includes a reporting architecture that greatly simplifies the process of developing custom reports by allowing the developer to specify the report requirements in a TaskDefinition XML document. The executor uses IdentityIQ's Forms API to generate the UI form for parameter specification and creates the report output based on column con figurations specified in the TaskDefinition. The XML specifies the report's Standard Properties values, the report-specific parameters, the columns that are available for the report, how the data is retrieved for inclusion on the report, and how the report results are laid out in both the detail and summary sections.

The standard report templates provide some good examples of how to define reports through XML. Excerpts from these standard templates are used in this chapter to illustrate how to configure custom reports. Many of the excerpts come from the Uncorrelated Accounts Report, a fairly simple example that can be used to explore the basics of defining a custom report.

It might be helpful to examine the full XML for these reports to see the tags' usage in context as they are referenced and excerpted in this document. The reports can be viewed through the debug pages as described in XML Representation of Reports and Instances, or the entire set of TaskDefinition objects can be exported to a file through the iiq console and explored in a text editor. The console export command to write the system's TaskDefinition objects to a file is “export taskDefs.xml TaskDefinition” (where “taskDefs.xml” is the name of the file to which the objects are exported). Note that the file contains all tasks including reports because no filter available on the export command to select only a subset of objects of a given type.

Report as a TaskDefinition

In IdentityIQ, a report is essentially executed as a specialized task. The root element of a report is a <TaskDefinition> element.

```
<TaskDefinition executor="sailpoint.reporting.LiveReportExecutor" name="Uncorrelated
Accounts Report" progressMode="Percentage" resultAction="Rename" subType="Identity
and User Reports" template="true" type="LiveReport">
```

The type attribute for the TaskDefinition indicates that this is a report definition, and the executor specifies which class processes this task definition to run the report. The attributes of the TaskDefinition object and their purposes are described in the table below.

Table 88—TaskDefinition Type Attributes

Attribute	Usage
executor	Indicates the class to run the report. In previous version of IdentityIQ, custom reports required a custom report executor. The new architecture includes the “sailpoint.reporting.LiveReportExecutor”, which is always specified as the executor for any report of type “LiveReport,” including custom reports.

Table 88—TaskDefinition Type Attributes

Attribute	Usage
name	Name of the report template; shown on the **Reports** list as the report's **Name** **NOTE: When templates are edited, they must be saved as customized report instances, and the name value across all report templates and report instances must be unique. Therefore, the name attribute for a template is not displayed as the Name field's value on the Edit Report window's Standard Properties page since instances cannot be saved with this same name. The value entered in the Standard Properties page's Name field becomes the name value of the TaskDefinition XML for that instance.**
progressMode	Specifies how the executor updates progress while the report is being executed; most reports use Percentage. Possible values are: **None** — executor doesn't update progress (same as null, or not specifying) **String** — executor periodically updates progressString property of the result during execution **Percentage** — executor periodically updates the progress and percentageComplete properties of the result during execution
resultAction	Specifies the **PreviousResultAction** (shown on the Standard Properties page) - states how to manage the results from previous runs of this report when it is executed again. Possible values are: **Rename** — rename old report results by appending a numeric value (for example, Uncorrelated Accounts Report - 2) **RenameNew** — rename new report results by appending a numeric value **Cancel** — do not run the report when old report results still exist for the report (displays an error message indicating that a result from a previous execution of the report still exists) **Delete** — delete old report results when the report is executed again
subType	Indicates the report category to which this report belongs (sub-categories within the **Reports** and **My Reports** tabs); can be one of the out-of-the-box subTypes or a custom subType
template	Boolean indicating whether this is a report template (appears on the **Reports** tab) or a customized report instance (appears on **My Reports** tab); new custom reports should be set up as template=”true”
type	All reports using the new reporting architecture, including new custom reports, are of type “LiveReport”; pre-6.0 reports are of type “Report”

Note: **When a report XML document is imported into IdentityIQ, two attributes - created and id - are generated by the system and saved as part of the TaskDefinition; these should not be specified in the XML by the report developer.**

Elements within TaskDefinition

The TaskDefinition contains several nested elements that are used to define important information for the report. The TaskDefinition always contains an Attributes map and a Signature and usually contains a Description element and a list of RequiredRights.

Attributes Map

The **Attributes** map minimally must contain the report definition that described in the next section..

```
   <Attributes>
      <Map>
        <entry key="report">
          <value>
            <LiveReport title="Uncorrelated Accounts Report">
                …
            </LiveReport>
          </value>
        </entry>
      </Map>
    </Attributes>
```

Other optional attributes in the attribute map include emailIdentities, reportSortBy, reportGroupBy, disableSummary, and disableDetail, as described in the Standard Forms for Report Specification section.

Signature

The Signature contains a map of Input attributes that names all of the parameters that can be specified for the report.

```
<Signature>
    <Inputs>
      <Argument multi="true" name="correlatedApps" type="Application">
        <Description>rept_input_uncorrelated_ident_report_correlated_apps
        </Description>
        <Prompt>report_input_correlated_apps</Prompt>
      </Argument>
      <Argument name="resultScope" type="Scope">
        <Description>rept_input_result_scope</Description>
      </Argument>
      <Argument multi="true" name="emailIdentities" type="Identity">
        <Description>rept_input_email_recips</Description>
      </Argument>
    </Inputs>
 </Signature>
```

When a customized report instance with pre-populated parameters is saved, those parameters are saved as a part of the instance's TaskDefinition in its Attributes map. In this example, a list of Applications and a list of Email Recipients have been saved for this report instance.

```
<TaskDefinition created="1344453735712" id="4028460238edaba4013907aff5200ec9"
modified="1344867911170" name="Uncorrelated Accounts" resultAction="Rename"
subType="Identity and User Reports" type="LiveReport">
 <Attributes>
    <Map>
       <entry key="correlatedApps">
        <value>
          <List>
            <String>4028460238edaba401372767b6eb0d70</String>
            <String>4028460238edaba40138edcfcd1102d2</String>
          </List>
        </value>
      </entry>
      <entry key="disableDetail" value="false"/>
      <entry key="disableSummary" value="false"/>
      <entry key="emailIdentities">
        <value>
          <List>
            <String>4028460238edaba40138edb3571e000d</String>
          </List>
        </value>
      </entry>
      <entry key="reportColumnOrder" value="username, firstName, lastName"/>
    </Map>
 </Attributes>
```

They are passed from the customized report instance to the associated report template at run-time through the taskDefinition input arguments specified in the <Signature> for the report.

Every report template's Signature should include input arguments for resultScope and emailIdentities, since these automatically appear on the Standard Properties window and are available for a user to specify on all reports. All other input arguments are report-specific and the remainder of the arguments in a custom report is specific to that report. The list of arguments should match the set of fields available for parameter specification on the report's Form. Report-specific arguments should include a name and type as attributes on the <Argument> element. If the argument is multi-valued, it should also include the attribute “multi=”true”.”

Application arguments can include a Description and Prompt element. When a custom form (in a <ReportForm> element) has been specified, these are ignored and can be omitted. However, if no custom form is specified, these report-specific input arguments are automatically rendered on a form page (titled Report Options) using the Prompt value as the field label and the Description value as the tool tip for the field. Both of these values can be specified as strings or as localizable message keys.

```
<Argument multi="true" name="correlatedApps" type="Application">
 <Description>rept_input_uncorrelated_ident_report_correlated_apps
 </Description>
 <Prompt>report_input_correlated_apps</Prompt>
```

The Account Group Members report is an example of a report that relies on this automatic form rendering for its report-specific filter options.

Description

The Description element is displayed as the Description for the report on both the Reports and My Reports lists and on the Edit Report window.

```
<Description>A detailed view of the uncorrelated user accounts in the
system.</Description>
```

Required Rights

The RequiredRights element specifies what system right(s) a user must have to be able to see and execute the report. The required rights are specified as references to one or more SPRight objects.

```
<RequiredRights>

  <Reference class="sailpoint.object.SPRight" id="4028460238ed9b8e0138ed9bc59d0054"
name="FullAccessUncorrelatedIdentitiesReport"/>

</RequiredRights>
```

Report Definition

The report, including its custom UI form or forms, its query specification, and its results contents and layout, is specified as a part of the TaskDefinition's attributes map with the attribute key “report”. The value for this attribute is a <LiveReport> element.

```
<Attributes>

    <Map>

      <entry key="report">

        <value>

          <LiveReport title="Uncorrelated Accounts Report">
```

Elements within <LiveReport> determine the report filters that can be specified by a report user, the query used to retrieve the data for the report, the layout of the report detail grid, and the contents and layout of the report's summary table and chart. The nested elements within <LiveReport> are:

Table 89—LiveReport Nested Elements

Element	Description
ReportForm	Determines how the report-specific parameters sections are presented to the user in the Edit Report window (see ReportForm: Collecting Report-Specific Parameters)
DataSource	Specifies how the data for the report details is retrieved from the database (see DataSource: Retrieving Report Data)
Columns	Lists columns available for inclusion in the report detail grid; also used in conjunction with DataSource to determine which data elements are retrieved in the query (see Columns/ReportColumnConfig: Report Grid Presentation)

Table 89—LiveReport Nested Elements

Element	Description
ReportSummary	Describes the Summary section of the report - information included, query to retrieve it, layout and labels for presentation of it (see ReportSummary: Summary Table)
Chart	Defines the graph or chart displayed in the Summary section for the report (see Chart: Report Graph)

Each of these elements is explored in detail in the next sections.

ReportForm: Collecting Report-Specific Parameters

Most reports allow users to specify filters that constrain the contents of the generated report. Report-specific parameters are collected from the report user through a custom form, referenced through a ReportForm element in the report definition. The form must be specified as a separate XML document and imported into IdentityIQ. The Form object is described in the Report Forms section at the end of this document.

The ReportForm element references the form like this:

```
<ReportForm>

<Reference class="sailpoint.object.Form" id="4028460238edaba40138edb36b330010"
name="Uncorrelated Account Report Custom Fields"/>

</ReportForm>
```

Standard Forms for Report Specification

The referenced ReportForm is presented to the user in the Edit Report window between the two standard form pages that are part of every report's specification: Standard Properties and Report Layout. Those two standard pages are rendered based on a Form object called Report Skeleton using values specified in the report's TaskDefinition XML. These tables indicate which TaskDefinition elements and attributes determine the values for fields on the Standard Properties and Report Layout pages.

Standard Properties Field	TaskDefinition Source
Name	If editing a customized report instance, <TaskDefinition> name attribute `<TaskDefinition name="Uncorrelated Accounts - Financials" … >` If creating a new report instance based on a template, none (not populated)
Previous Result Action	`<TaskDefinition> resultAction attribute<TaskDefinition name="Uncorrelated Accounts Report" resultAction="Rename" … >`
Description	<Description> element
Scope*	resultScope entry in Attributes map (value contains ID of selected scope) `<entry key="resultScope" value="2c9082ee38e813a20138e934eb210146"/>`

Standard Properties Field	TaskDefinition Source

Email Recipients*	emailIdentities entry in Attributes map (List contains Identity ID values)
	
	`<entry key=”emailIdentities”>`
	` <value>`
	`    <List>`
	`       <String>4028460238edaba40138edb35653000b</String>`
	`    </List>`
	` </value>`
	`</entry>`
	
	Usually specified in instance XML instead of template XML

Allow Concurrency	`<TaskDefinition> concurrent attribute (concurrent=”true”)<TaskDefinition concurrent="true" name="Uncorrelated Accounts Report" … >`

Require Signoff*	<SignoffConfig> element
	
	` <SignoffConfig>`
	`   <WorkItemConfig created="1344962495866" escalationStyle="none" id="4028460238edaba401392603057a1464">`
	`      <NotificationEmailTemplateRef><Reference class="sailpoint.object.EmailTemplate" id="4028460238ed9b8e0138ed9bd8690106" name="Default Report Template"/>`
	`      </NotificationEmailTemplateRef>`
	`      <Owners><Reference class="sailpoint.object.Identity" id="4028460238edaba40138edb36b33016d" name="Aaron.Nichols"/>`
	`      </Owners>`
	`    </WorkItemConfig>`
	`</SignoffConfig>`

Report Layout Field	TaskDefinition Source
Sort By*	reportSortBy entry in Attributes map `<entry key="reportSortBy" value="accountGroupDisplayName"/>`

Sort Ascending*	reportSortAsc entry in attributes map
	
	`<entry key="reportSortAsc">`
	`   <value>`
	`     <Boolean>true</Boolean>`
	`   </value>`
	`</entry>`

Group By*	reportGroupBy entry in Attributes map `<entry key="reportGroupBy" value="application"/>`

Report Layout Field	TaskDefinition Source
Columns	<ReportColumnConfig> header attributes; hidden="true" attribute places column in left pane - available but not included on report detail by default `<ReportColumnConfig field="accountGroupName" header="rept_app_account_grp_memb_col_name" property="value" sortable="true" />`
Disable Report Summary Display*	disableSummary entry in Attributes map `<entry key="disableSummary" value="true"/>`
Disable Report Detail Display*	disableDetail entry in Attributes map `<entry key="disableDetail" value="true"/>`

* These fields are typically specified through the UI for customized instances of reports and saved into the My Reports instances' attributes map, rather than being specified in the report template XML. However, they can be specified in the template XML if they apply to the report's default configuration.

Although most reports do include a custom form, it is not required. When one is not specified, the Edit Report window still displays the Standard Properties and Report Layout pages; the Identity Status Summary report shows an example of this.

DataSource: Retrieving Report Data

The data shown in the detail section of the report is retrieved through a query that is built based on a combination of the <DataSource> specification and the <Columns> element. In general, a query is specified in three parts: Select, From, and Where. The Select portion (the columns list) is specified through the <Columns> element in the report definition - specifically, the <ReportColumnConfig>s listed within <Columns> element. The From and Where clauses are specified through the <DataSource> element.

There are three available datasource types: Filter, Java, and HQL. The simplest of these three is the Filter datasource, though various options available with this datasource type make it quite powerful and flexible. The other two are available for more complex report data retrieval needs, and Java is likely to be used as the datasource more often in HQL in those cases. Each of these three datasource types is discussed next.

Filter DataSource

A filter datasource executes a projection query to retrieve the data required by the ReportColumnConfigs specified for the report. It employs the SailPoint Filter object to specify the query. The object whose data is being queried is specified as the objectType for the DataSource, and the DataSource type is specified as “Filter”.

```
<DataSource objectType="sailpoint.object.Link" type="Filter">
```

Note: If the objectType is one of the top-level classes in the IdentityIQ object model (for example, the set of objects that can be exported from the iiq console or retrieved directly in from the debug pages), the fully-qualified class name is not required for this attribute. For example, Identity can be specified here as objectType=”Identity”. However, the fully-qualified name (for example, sailpoint.object.Identity) is always acceptable, even for the top-level classes, so when in doubt, specify the fully-qualified name.

This is an example of a filter <DataSource> and its <Columns> specification:

```
<LiveReport title="Uncorrelated Accounts Report">
 <DataSource objectType="sailpoint.object.Link" type="Filter">
   <QueryParameters>
<Parameter argument="correlatedApps" property="application.id"/>
       <Parameter defaultValue="false" property="identity.correlated"
valueClass="Boolean"/>
       <Parameter defaultValue="false" property="application.authoritative"
valueClass="Boolean"/>
       <Parameter defaultValue="false" property="application.logical"
valueClass="Boolean"/>
   </QueryParameters>
 </DataSource>
 <Columns>
<ReportColumnConfig field="username" header="rept_uncorrelated_ids_grid_username"
property="nativeIdentity" sortable="true" />
<ReportColumnConfig field="firstName" header="rept_uncorrelated_ids_grid_firstName"
property="identity.firstname" sortable="true" />
<ReportColumnConfig field="lastName" header="rept_uncorrelated_ids_grid_lastName"
property="identity.lastname" sortable="true" />
<ReportColumnConfig field="applicationName"
header="rept_uncorrelated_ids_grid_appName" property="application.name"
sortable="true" />
 </Columns>
```

The search criteria, making up the "where" clause for the search, are specified through one or more of several query-related elements: Query, QueryParameters, and QueryScript. Joins, sorts and groupBy columns can also be specified as needed for the query.

QueryParameters

The <QueryParameters> element is used most often. QueryParameters is a map of argument values used to create the queryOptions object that controls the search. They can be specified based on report arguments, hard-coded values, or calculated values. QueryParameters contains a list of <Parameter> elements, each of which defines one of the criteria. These <Parameter>s are "anded" together to make the where clause.

```
<QueryParameters>
   <Parameter argument="correlatedApps" property="application.id"/>
   <Parameter defaultValue="false" property="identity.correlated"
valueClass="Boolean"/>
   …
</QueryParameters>
```

There are several different options for specifying parameters in a set of QueryParameters. These options are described below, illustrated with example Parameters. Most of these examples (except where noted) were taken from the Entitlement Owner Access Review Live Report which queries against the **sailpoint.object.CertificationItem** object, so all of these parameters relate to that object.

- **Referencing a report argument**: generally processed as “property = argument”; this parameter looks for certificationItems with a parent.certification.certificationGroups.id value in the report argument “certificationGroups”

```
<Parameter argument="certificationGroups"
property="parent.certification.certificationGroups.id"/>
```

Note: When arguments are multi-valued, parameters based on them are automatically evaluated with “in” rather than “equals”.

- **Specifying a default value**: generally processed as “property = argument or defaultValue (if argument is null)”; this parameter looks for CertificationItems with a parent.certification.type equal to the report argument “type”; if none is provided, it defaults to the type “DataOwner”

```
<Parameter argument="type" defaultValue="DataOwner"
property="parent.certification.type"
valueClass="sailpoint.object.Certification$Type"/>
```

Note: This example also illustrates usage of the valueClass attribute; this attribute is not necessary for string or object comparisons but is for other types, such as enumerations (such as Type in this example), Booleans, Dates, Lists, etc.

- **Specifying a hard-coded value**: an attribute can also be hard coded to be evaluated against the defaultValue by not including an argument, as shown in this parameter from the Uncorrelated Accounts Report. This is processed as “property = defaultValue”, in this case cast as valueClass (not required for strings).

```
<Parameter defaultValue=”false” property=”identity.correlated”
valueClass=”boolean”/>
```

- **Specifying different operations**: this example illustrates how to create evaluation conditions other than equals (or in) relationships; operation can be specified as GT, GE, LT, or LE (greater than, greater than or equal to, less than, less than or equal to)

```
<Parameter argument="createStartDate" operation="GT"
property="parent.certification.created"/>
```

- **Using a ValueScript**: processed as "property = return value from ValueScript"; this parameter performs processing based on the argument to return a different value that should be used in the criterion; this example uses a ValueScript to get the application name that corresponds to the applicationID in the "application" report argument; in a ValueScript, the argument is accessed through the variable name "value".

```
<Parameter argument="applications" property="parent.application">
   <ValueScript>
      <Source>
         import sailpoint.object.*;
         import sailpoint.api.ObjectUtil;
         if (value != null){
            return ObjectUtil.convertIdsToNames(context, Application.class,
value);
         }
         return null;
      </Source>
   </ValueScript>
</Parameter>
```

Note: Since object references are stored in the customized report instance XML (and passed to report input arguments) as ID values and many comparisons need to be done based on name, this convertIdsToNames() utility method is frequently used in ValueScripts in the standard reports.

- **Using a QueryScript**: used to specify any custom filter and add it into the queryOptions object that is used in the datasource filter; parameters using a QueryScript do not need to specify a property because the queryScript overrides any property on the parameter; the argument specified on the parameter can be accessed within the script through the "value" variable

  Group and populations are stored in groupDefinitions objects as a filter, so this example (from the Identity Forwarding Report) shows how a group or population selected as a report parameter is built into the datasource filter through a QueryScript.

```
<Parameter argument="groupDefinitions">
   <QueryScript>
      <Source>
         import sailpoint.object.*;
         import sailpoint.reporting.*;
         Filter f = ReportingLibrary.getGroupDefinitionFilter(context, value, 
false);
         if (f!=null) {
            queryOptions.addFilter(f);
         }
         return queryOptions;
      </Source>
   </QueryScript>
</Parameter>
```

- **ValueRule and QueryRule**: These two can be specified in place of ValueScript and QueryScript, respectively, to encapsulate the beanshell of a script into a reusable rule. (These two examples were not pulled from a standard report; they represent the appropriate syntax if the reports using the ValueScript and QueryScript specified above had encapsulated those scripts into rules.)

```
<Parameter argument="applications" property="parent.application">
   <ValueRule>
      <Reference class="sailpoint.object.Rule" 
id="4028460238ed9b8e0138ed9beff9090f" name="App Value Rule"/>
   </ValueRule>
</Parameter>

<Parameter argument="groupDefinitions">
   <QueryRule>
      <Reference class="sailpoint.object.Rule" 
id="4028460238ed9b8e0138ed9beff90900" name="Group Query Rule"/>
   </QueryRule>
</Parameter>
```

Query

Another way to specify the filter contents is though a <Query> element. The contents of Query element are specified as a filter string and can only specify hard-coded criteria with no variable substitution (i.e. report arguments cannot be specified within a Query element). Query allows the specification of "or" criteria, as shown in the example below:

```
<Query>IdentityEntitlement.name=="assignedRoles" ||
IdentityEntitlement.name=="detectedRoles"</Query>
```

Query and QueryParameters can be specified for the same DataSource. When both are specified, the Query filter and the Parameter filters are "anded" together to create the final where clause.

QueryScript

QueryScript creates a filter string through a beanshell script. It is designed so it can append additional criteria, including those requiring variable substitution, onto a Query element's contents. The script has access to the string value of the Query element (in a string variable called "query") and must explicitly append the additional criteria to it; otherwise, the original query string is overwritten with the QueryScript's return value. The QueryScript shown below actually comes from an HQL datasource report (the Account Group Membership Totals Report), but the QueryScript syntax is the same for all datasource types.

```
<QueryScript>
   <Source>
      import java.util.*;

      List applications = args.get("application.id");
      if (applications != null &amp;&amp; !applications.isEmpty()){
         query = query + " and application.id in(:application_id) ";
      }
      return query;

    </Source>
</QueryScript>
```

Join

When the search must access more than one object to process the filter criteria, a <Join> element is required to connect the objects properly. One or more Joins can be specified for a single datasource.

For example, the Identity Roles Report displays the roles that each Identity is assigned. Most of the available filters for the report apply to the Identity object, but the role assignment is recorded on the IdentityEntitlement object, linked to the Identity object by the Identity ID. The Join element specifies that connection. The property is the value on the primary object (the DataSource objectType) and the joinProperty specifies the connection attribute on the second object.

```
<DataSource objectType="Identity" type="Filter">
   <Join joinProperty="IdentityEntitlement.identity.id" property="id"/>
   <Query>IdentityEntitlement.name=="assignedRoles" ||
IdentityEntitlement.name=="detectedRoles"</Query>
   <QueryParameters>
      <Parameter argument="identities" property="id"/>
      …
   </QueryParameters>
</DataSource>
```

OptionsRule or OptionsScript

The final elements available on a filter datasource are an OptionsRule or OptionsScript. These can be used to make modifications to the QueryOptions before the query is run; they can also replace the rest of the query specification (for example, eliminating the need for a Query, QueryParameters, QueryScript or Join element) by simply constructing the whole queryOptions in the rule or script.

Only one of these can be specified (the rule overrides the script if both are provided). The OptionsRule or OptionsScript is passed a SailPoint Context called “context”, a queryOptions called “options” and an argument map called “args”. Options contains the entire set of query criteria specified in any of the other elements (Query, QueryScript, QueryParameters, Join) and args contains the TaskDefinition argument map. The rule or script should append any additional custom queryOptions to options and return it.

```
 <OptionsScript>
    <Source>
       import java.util.*;
       import sailpoint.object.*;

       //code to add components to queryOptions goes here. e.g.: this would
       // Apply to an Identity objectType and would get only Identities whose
       // Manager is the Identity selected in the manager filter (typically,
       // an optionsScript or optionsRule would be used for a more complex

       Filter myFilter = Filter.eq("manager.id", args.get("manager.id");
       options.addFilter(myFilter);

       return options;

     </Source>
 </OptionsScript>
```

An OptionsRule is specified as a reference to a Rule object:

```
<OptionsRule>
    <Reference class="sailpoint.object.Rule" id="4028460238ed9b8e0138ed9beff90900"
name="MyReport Options Rule"/></OptionsRule>
```

Java DataSource

A Java datasource is the next most commonly used report datasource type. The XML to specify this is fairly simple and straightforward; the java class it calls can be as simple or as complex as is required to generate the desired report contents.

The java datasource class must implement the **sailpoint.reporting.datasource.JavaDataSource** interface, as described in the IdentityIQ javadocs. This interface defines all the methods that must be coded. All attributes in the taskDefinition attribute map (including all input attributes from the Signature) are passed to the Java class in an arguments map.

The <DataSource> element in the XML specifies these attributes:

DataSource Attribute	Usage
dataSourceClass	The fully qualified java class name
objectType	The primary object against which searches are performed in the java code
type	Java (tells the report executor this is a Java Datasource)
defaultSort	Optional field; sorts the returned data by the named field if no sort column is specified through the UI or taskDefinition attributes map

Note: Many of the standard reports were written with a Java Datasource and several examples of this syntax are available. Most of the standard reports use a QueryParameters element to pass data to the DataSource, which allowed the report writer to take advantage of the reportHelper class in the reporting architecture to reuse existing code. However, this is not strictly necessary and isnot commonly done in the field. Because the entire taskDefinition attributes map, including all input attributes from the <Signature>) is passed to the java class in an arguments map, they do not need to be specified as QueryParameters. The class can build the QueryOptions object needed to retrieve the data without passing the values through QueryParameters.

HQL DataSource

An HQL datasource is used in rare circumstances but is available for implementers who need to execute queries that hit Hibernate directly. This should only be used when the report developer is very knowledgeable about HQL. The HQL query must be custom written by the report developer.

Like the Filter datasource, the HQL datasource can specify its query using these types of nested elements: Query, QueryScript, and QueryParameters. The Query and QueryParameters elements function somewhat differently in an HQL datasource, though, so it is important to understand the way they are processed.

The Account Group Membership Totals Report provides an example of an HQL datasource.

```
<LiveReport title="Account Group Membership Totals Report">
   <DataSource type="Hql">
      <Query>from ManagedAttribute m where group=true</Query>
      <QueryParameters>
          <Parameter argument="application" property="application_id"/>
      </QueryParameters>
      <QueryScript>
         <Source>
            import java.util.*;

            List applications = args.get("application.id");
            if (applications != null &amp;&amp; !applications.isEmpty()){
               query = query + " and application.id in(:application_id) ";
            }
            return query;
          </Source>
        </QueryScript>
    </DataSource>
    <Columns>
          <ReportColumnConfig field="accountGroupName"
header="rept_app_account_grp_memb_col_name" property="value" sortable="true"/>
          <ReportColumnConfig field="accountGroupDisplayName"
header="rept_app_account_grp_display_name" property="displayName" sortable="true"/>
      <ReportColumnConfig field="application" header="rept_app_account_grp_memb_app"
property="application.name" sortable="true"/>
          <ReportColumnConfig field="total"
header="rept_app_account_grp_memb_col_members" property="(select count(*) from
IdentityEntitlement ie where ie.value = m.value and ie.application = m.application and
ie.name = m.attribute and ie.aggregationState = &apos;Connected&apos;)"/>
    </Columns>
</LiveReport>
```

In an HQL datasource, the <Query> element must specify the From clause for the query. The objectType is not required for an HQL datasource and is ignored if it is provided.

Query

The Query element can also specify some or all of the where clause. As on a Filter DataSource, the Query element can specify any hard-coded attribute evaluations (i.e. no variable substitution available) and multiple conditions can be specified with “and” or “or” relationships.

```
      <Query>from ManagedAttribute m where group=true</Query>
```

QueryScript

The HQL DataSource <QueryScript> element works just like the Filter Datasource QueryScript. It contains beanshell that returns a filter string (appending to the Query's string and returning the combined string value). However, the difference in QueryParameter processing changes the way variables are processed in the script. The queryScript has access to the task argument map (in its “args” variable), so conditional processing can be done on those arguments in determining how to build the filter string. However, the contents of those variables do not need to be built into the actual query string in the queryScript; they can be referenced as variable names that are passed to the search through QueryParameters. In an HQL datasource, the search is performed based on the query string built in the query and queryScript elements; the parameters specified as QueryParameters are passed to the search method along with that query string and are substituted into the query where variable names are found.

In the example below (from the Account Group Membership Totals Report), the QueryScript examines the application.id value from the args list and if it is non-null, it appends “and application.id in (:application_id) “ to the query string. The QueryParameter application_id allows the list of applications from the task argument list to be substituted for the :application_id variable in that query string when the search is executed.

```
<QueryParameters>
    <Parameter argument="application" property="application_id"/>
</QueryParameters>
```

```
<QueryScript>
    <Source>
       import java.util.*;

       List applications = args.get("application.id");
       if (applications != null &amp;&amp; !applications.isEmpty()){
          // :application_id
          query = query + " and application.id in(:application_id) ";
       }
       return query;

    </Source>
</QueryScript>
```

QueryParameters

As explained in the QueryScript section above, the QueryParameters in an HQL datasource do not make up filter components in their own right but instead provide variables for substitution into the query string at the time the search is executed.

The Parameter elements within the QueryParameters for an HQL datasource is usually only specified with an argument and a property. The property is the variable name used in the query string and the argument is the argument map key in which the value to be used in the search is stored. A defaultValue or a valueScript (as described in the Filter datasource's QueryParameters section) can also be used to provide the value for the property, if desired. The Parameter's QueryScript option (which returns a QueryOptions object) cannot be used for an HQL datasource, as it does not provide a value for substitution; HQL datasources do not use a QueryOptions object in their searches.

ReportColumnConfigs

Just as with the other report types, the ReportColumnConfigs within the report's <Columns> element specify the attributes to retrieve from the query for display in the report detail grid - the “Select” portion of the query. The property attributes name the fields to retrieve. The final ReportColumnConfig - the “total” column - in the Account Group Membership Totals Report shows an example of how to include a sub query in the HQL select clause. This provides additional levels of flexibility in reflecting data on the report. A calculated field like this cannot be marked as sortable.

```
<ReportColumnConfig field="total" header="rept_app_account_grp_memb_col_members"
property="(select count(*) from IdentityEntitlement ie where ie.value = m.value and
ie.application = m.application and ie.name = m.attribute and ie.aggregationState =
&apos;Connected&apos;)"/>
```

Columns/ReportColumnConfig: Report Grid Presentation

The <ReportColumnConfig> elements within the <Columns> element specify which values should be returned from the query and also define how those values are presented in the report grid.

```
<Columns>
```

```
<ReportColumnConfig field="username" header="rept_uncorrelated_ids_grid_username"
property="nativeIdentity" sortable="true" />

<ReportColumnConfig field="firstName" header="rept_uncorrelated_ids_grid_firstName"
property="identity.firstname" sortable="true" />

<ReportColumnConfig field="lastName" header="rept_uncorrelated_ids_grid_lastName"
property="identity.lastname" sortable="true" />

<ReportColumnConfig field="applicationName"
header="rept_uncorrelated_ids_grid_appName" property="application.name"
sortable="true" />

 </Columns>
```

Attributes of ReportColumnConfig include:

Attribute	Usage
field	Unique name for the report column in this report
header	Column label to use in the report body; can be a string or a localizable message key
property	Object property from which the data is pulled; this value is used in the query specification
sortable	Boolean value indicating whether the report body should be sortable by this column; determines whether the column is selectable in the Sort By and Group By fields in the report specification and whether the report can be sorted by this column in preview mode
hidden	Boolean value indicating whether the column should be omitted from the report grid by default. Columns marked as hidden (hidden=”true”) appear in the left-side of the Columns list on the report template's Report Layout page, which makes them available for inclusion. However, by default they are not included on the report. Any report instances that were configured to display these fields in the report grid override this hidden attribute by including the column name in their reportColumnOrder attribute, causing the column to appear in the report output regardless of this attribute's value.
ifEmpty	Optional property to use if the value of the object property is null or empty;See Entitlement Owner Access Review Live Report's accountName field for an example: `<ReportColumnConfig field="accountName" header="rept_data_owner_col_account_name" ifEmpty="exceptionEntitlements.nativeIdentity" property="exceptionEntitlements.displayName"  sortable="true" width="110"/>`
subQueryKey	Used for multi-valued properties to show the values as a list of comma-separated values instead of multiple rows in the report. Specifying a subQueryKey automatically renders the column as a subquery that selects the property from the dataSource objectType matching on the subsQueryKey attribute.An example exists in the Manager Access Review Live Report's tags field: `<ReportColumnConfig field="tags" header="rept_cert_col_tags" property="parent.certification.tags.name" subQueryKey="id" width="110"/>`

Attribute	Usage

sortExpression	A set of fields by which the data should be sorted instead of sorting by the selected column. This attribute allows a column that is not sortable to sort the data by columns related to the selected column.
	See the following example of the permission column on the Account Group Permissions Access Review Live Report.
	
	`<ReportColumnConfig field="permissions" header="rept_cert_col_account_group_permission" property="exceptionEntitlements"sortExpression="exceptionApplication,exceptionPermissionTarget,exceptionPermissionRight" sortable="true" width="110">`
	` <RenderScript>`
	`    <Source>`
	`       return sailpoint.api.EntitlementDescriber.summarize(value);`
	`    </Source>`
	` </RenderScript>`
	`</ReportColumnConfig>`

scriptArguments	A CSV list of additional properties to pass to a column's RenderScript; see the status field from the Policy Violation Report for an example. The renderScript then accesses these values through its scriptArgs variable (as shown in this example).
	
	`<ReportColumnConfig field="status" header="rept_viol_grid_col_status" property="status" scriptArguments="identity,policyName,constraintName,created" sortable="true" width="110">`
	`     <RenderScript>`
	`       <Source>`
	`          import sailpoint.object.*;`
	`          …`
	`          String identityId = scriptArgs.get("identity").id;`
	`          …`
	`       </Source>`
	`     </RenderScript>`

valueClass	Defines the class for the property so it can be displayed appropriately; omitted for string values
skipLocalization	Indicate that the column contains reserved words that should not be translated. Examples include Names or Account names that could contain reserved keywords.

In the standard reports, the only time the “hidden” attribute is used on a ReportColumnConfig is when the column is added to the available set by an ExtendedColumnScript or ExtendedColumnRule (as described in Extended Column Script or Rule). Generally, if a column is relevant to a report, it is displayed on the report by default, though it can be removed from the detail grid by a user if they do not wish to see that data on their customized version of the report.

Note: Strings and Java constants specified in ReportColumnConfig attributes are evaluated first as message keys for automatic localization; if they do not match a defined message key, the given string value is used.

RenderScript and RenderRule

If the value returned from the query needs to be manipulated into a more user-friendly format for display on the report, this can be accomplished with a RenderScript or RenderRule. A RenderRule is used to encapsulate the beanshell into a reusable rule - useful when the same manipulation might apply to several reports. A RenderScript specifies the beanshell inline within a <Source> element. The column's property attribute is passed into the script in the variable “value”.

This example RenderScript (taken from the Revocation Live Report) displays a different localized message key depending on whether the action.remediationCompleted flag is true or false so that the report column shows an easier-to-interpret “Status” instead of a True/False flag.

```
<ReportColumnConfig field="status"
header="rept_remediation_progress_grid_col_status"
property="action.remediationCompleted" sortable="true" width="110">

   <RenderScript>

      <Source>

         import sailpoint.tools.Message;

         import sailpoint.web.messages.MessageKeys;

       return value == true ? Message.localize(MessageKeys.WORK_ITEM_STATE_FINISHED)
: Message.localize(MessageKeys.WORK_ITEM_STATE_OPEN);

      </Source>

         </RenderScript>

</ReportColumnConfig>
```

A RenderRule would be specified like this:

Rendered columns are sorted by the property attribute, not by the displayed value, so the order of rows might not appear alphabetical by the display value. At a minimum, sorting by the column groups all of the rows with the same column value together. Some properties might not be sortable, such as a property that is an object. These columns should be marked as sortable="false" even though the displayed value might seem sortable. Alternatively, a sortExpression can be specified to drive data sorting for these columns.

```
<ReportColumnConfig field="status"
header="rept_remediation_progress_grid_col_status"
property="action.remediationCompleted" sortable="true" width="110">

   <RenderRule>

     <Reference class="sailpoint.object.Rule" id="4028460238ed9b8e0138ed9bf61300de"
name="Status Message RenderRule"/>

   </RenderRule>

</ReportColumnConfig>
```

Initialization Script or Rule

The initialization script and rule allow the report developer to customize a report to address an installation's unique reporting requirements. These scripts/rules are fairly open-ended and should generally be considered tools for expert-level report creation.

Most often, initialization scripts and rules are used to customize the forms presented to the user for filter specification. For example, several standard reports use an initialization rule to build dynamic forms to present all of the installation's configured Identity attributes - both standard and extended - as filter options on some forms. Another form customization usage might be to change the set of filters available based on other filter selections; for example, a report might present a “privileged” account filter option only when the application selected for the “Application” filter has privileged accounts.

An InitializationScript is specified inline within a <Source> element:

```
<InitializationScript>
   <Source>
     import sailpoint.object.*;
     import sailpoint.reporting.ReportingLibrary;
     … (initialization code goes here; see rule example below)
   </Source></InitializationScript>
```

An InitializationRule is specified as a rule reference with the code encapsulated in the named rule:

```
<InitializationRule>
    <Reference class="sailpoint.object.Rule" id="4028460238ed9b8e0138ed9bf6130000"
name="Identity Report Form Customizer"/>
</InitializationRule>
```

The rule shown below is used in several of the standard reports (such as the User Detail Report and Identity Roles Report) to customize a form based on the standard and extended Identity attributes configured for the installation. Similar rules exist to create custom forms for other reports.

```
<Rule language="beanshell" type="ReportCustomizer" name="Identity Report Form
Customizer">
 <Description>
    This rule populates a form with fields for the standard and extended identity
attributes.
  </Description>
  <Signature returnType="Map">
    <Inputs>
      <Argument name="locale">
        <Description>
          The current user's locale
        </Description>
      </Argument>
      <Argument name="report">
        <Description>
          The base report
        </Description>
      </Argument>
    </Inputs>
    <Returns>
    </Returns>
  </Signature>
  <Source>
     <![CDATA[
     import sailpoint.object.*;
     import sailpoint.reporting.ReportingLibrary;

     ObjectConfig identityConfig = ObjectConfig.getObjectConfig(Identity.class);
     // Add standard attributes to the form

     List standardAttributes = new ArrayList();

standardAttributes.add(identityConfig.getObjectAttributeMap().get("firstname"));
     standardAttributes.add(identityConfig.getObjectAttributeMap().get("lastname"));

standardAttributes.add(identityConfig.getObjectAttributeMap().get("displayName"));
     standardAttributes.add(identityConfig.getObjectAttributeMap().get("email"));
     standardAttributes.add(identityConfig.getObjectAttributeMap().get("manager"));
     standardAttributes.add(identityConfig.getObjectAttributeMap().get("inactive"));

     ReportingLibrary.addAttributes(context, report, Identity.class,
standardAttributes, null, "Identity Attributes", locale);

     // add extended attributes to the form (multi-valued and regular)

     List extendedAttrs = new ArrayList();
     for(ObjectAttribute att : identityConfig.getSearchableAttributes()){
       if (!att.isStandard())
         extendedAttrs.add(att);
     }

     for(ObjectAttribute att : identityConfig.getMultiAttributeList()){
         extendedAttrs.add(att);
     }

     ReportingLibrary.addAttributes(context, report, Identity.class, extendedAttrs,
null, "Identity Extended Attributes", locale);
```

```
    ]]>
</Source>
</Rule>
```

The methods in the ReportingLibrary (like the one used in this example rule) are documented in the IdentityIQ Javadocs. The addAttributes method, for example, does the following:

1. Determines the form page where the attributes should be displayed (selects by section name, creates a new page based on the section name if not found, or selects the first page after Standard Properties if no section is specified)
2. Adds each attribute as an extended argument to the LiveReport object
3. Adds each attribute to the datasource QueryParameters list as a Parameter
4. Defines a Field object for each attribute and adds it to the Section

These methods can be used in custom report development, but note that it is possible that they could change in future versions of IdentityIQ, requiring those reports that rely on them to be revisited and modified. Alternatively, the code to add the attributes to the query parameters and form fields list can be explicitly written by the datasource developer.

Note: When an initialization script/rule is in place, if any of the functionality depends on the value of a specific form field, that field must be specified with the postBack attribute set to true (postBack= "true"); this causes the form to submit and reload when that value changes, and causes the initialization rule or script to execute again, picking up the new value for the field.

Signature Extended Arguments

When the initialization script or rule adds new fields to a report form, the values must be saved for any report instance for which they were specified, and they must be passed to the report at runtime to be used as report filters. This is done by adding them as extended arguments in the report definition; from there, they are automatically stored in the report instance's argument map when the report instance's TaskDefinition is saved. Even though these arguments do not exist in the report template's signature, they are generated at runtime by the initialization script and the values from the template's argument map are applied for the report's execution.

Note: This only works for these initialization-generated attributes; all static form fields must be explicitly specified in the report template's signature for them to be used in the report generation. Attributes that are included in the report instance's attribute map that do not exist in the report signature and are not generated by the initialization script or rule is not applied to the report as filters at runtime.

Extended Column Script or Rule

An extended column script or rule can be used to add additional columns to a form based on other attributes selected. For example, the script shown below adds application-attribute columns to the set of available columns based on the application selected on the form (for example, if an application has a “privileged” or “service” account attribute, these can be optionally included in the report output when that application is selected as a filter for the report while they would not be available if a different application that did not have these attributes were selected). The extendedColumnScript or Rule should return a list of ReportColumnConfig objects; these are automatically added to the Columns list as “hidden” columns - available for inclusion on the report but not included in the report detail grid by default.

This script comes from the User Account Attributes Report and is used to add columns to the report output based on which application is selected as a filter for the report.

```
<ExtendedColumnScript>
   <Source>

   import java.util.*;
   import sailpoint.reporting.*;
   import sailpoint.object.*;

   List newCols = new ArrayList();
   Map formValues = form.getFieldValues();
  if (formValues != null &amp;&amp; formValues.containsKey("application") &amp;&amp;
formValues.get("application") != null){
       newCols = ReportingLibrary.createApplicationAttributeColumns(context,
formValues.get("application"));
   }

   return newCols;

   </Source>
</ExtendedColumnScript>
```

An ExtendedColumnRule is specified as a rule reference with the code encapsulated in the named rule:

```
<ExtendedColumnRule>
    <Reference class="sailpoint.object.Rule" id="4028460238ed9b8e0138ed9bf6130000"
name="Application Extended Column Rule"/>
</ExtendedColumnRule>
```

Note: When an extended column script/rule is in place, the field on which its functionality depends must be specified with the postBack attribute set to true (postBack= "true"); this causes the form to submit and reload when that filter field's data value changes, causing the ExtendedColumnRule to fire and detect the required condition for displaying the columns.

When columns are added to the report as a result of this rule, they first appear as “hidden” columns - available for inclusion in the report output but not selected for it. While they are still hidden, they are not saved in the report instance's XML but are regenerated as hidden columns by this rule every time the report specification is edited. Once the user adds the columns to the report detail's column list, the columns are saved in the customized report instance's attributes map in the ReportColumnOrder element, prefixed with the associated application's ID; if the report is later edited to reference a different application, these columns are automatically deleted from the report.

Validation Script or Rule

A validation rule or script is used to validate the data entered on a report form. For example, if a value is required for a specific filter for the report to run, that field can be validated as being non-null by a ValidationRule or ValidationScript.

A Validation script contains the code inline, wrapped in a <Source> element.

```
<ValidationScript>
    <Source>
        import java.util.*;
        import sailpoint.reporting.*;
        import sailpoint.object.*;
        List messages = new ArrayList();
        … (validation code goes here - see rule example below)
        return messages;
    </Source>
</ValidationScript>
```

A validation Rule is called by reference:

```
<ValidationRule>
    <Reference class="sailpoint.object.Rule" id="4028460238ed9b8e0138ed9bf61300ff"
name="Privileged Access Report Validation Rule"/>
</ValidationRule>
```

This validation rule checks the field on the Priviledged Account Attributes form for a null (or empty) value; the report requires that a value be specified for this field, so an error message is displayed and the report does not run if this field does not pass this validation. A validation script or rule returns a list of messages; if the form passes validation, this list should be empty.

```
<Rule language="beanshell" type="ReportValidator" name="Privileged Access Report
Validation Rule">
   <Description>
      This rule validates the Privileged Access Report Form
    </Description>
    <Signature returnType="java.util.List">
      <Inputs>
        <Argument name="context">
          <Description>
            A sailpoint.api.SailPointContext object that can be used to query the
database if necessary.
          </Description>
        </Argument>
        <Argument name="report">
          <Description>
            The report object
          </Description>
        </Argument>
        <Argument name="form">
          <Description>
```

```
              The submitted sailpoint Form object.
            </Description>
          </Argument>
        </Inputs>
        <Returns>
          <Argument name="messages">
            <Description>
              A list of error messages.
            </Description>
          </Argument>
        </Returns>
      </Signature>
      <Source>
        <![CDATA[
         import java.util.*;
         import sailpoint.object.*;
         import sailpoint.tools.Message;
         List messages = new ArrayList();

         Form.Section section = form.getSection("Priviledged Account Attributes");
         boolean found = false;
         for(FormItem item : section.getItems()){
           Field field = (Field)item;
           if(field.getValue() != null && field.getValue() != "") {
             found = true;
           }
         }

         if (!found)
              messages.add(Message.localize("rept_priv_access_err_no_attr"));

         return messages;
        ]]>
    </Source>
  </Rule>
```

ReportSummary: Summary Table

The <ReportSummary> element describes the summary table in the summary section of the report.

The table header is specified in the title attribute.

```
<ReportSummary title="Uncorrelated Account Details">
```

The report summary has its own datasource; it does not use the same datasource as the report detail grid. The datasource for the report summary can be specified as a script or a rule. A script is most commonly used, but the datasource beanshell can be encapsulated in a rule for reusability if desired. Both are expressed as nested elements (<DataSourceScript> or <DataSourceRule>).

The DataSourceScript or DataSourceRule for the ReportSummary is passed these parameters:

DataSourceScript or Rule Parameters	Contents/Purpose
Context	A SailPoint Context object for executing the search
reportArgs	The TaskDefinition argument/attribute map
Report	The entire LiveReport report definition
baseHql	The from and where clause used in the report detail search if the report DataSource was an HQL datasource; null if DataSource type was not HQL
baseQueryOptions	The QueryOptions (specifying the “where” clause criteria) used in the report detail search if the report DataSource was a Filter datasource; null if Datasource type was not Filter

The summary table is usually built from data retrieved through one or more database queries that are specified and executed by the script or rule through the context (SailPointContext object), passing it a QueryOptions object populated with the necessary Filter objects. The example here illustrates how this is done.

```
<ReportSummary title="Uncorrelated Account Details">
    <DataSourceScript>
       <Source>

          import java.util.*;
          import sailpoint.tools.Util;
          import java.lang.Math;
          import sailpoint.object.*;
          import sailpoint.api.ObjectUtil;

          QueryOptions ops = new QueryOptions();
          ops.addGroupBy("correlated");

          String sources = "";
          // retrieve list of apps in reportArgs argument map; add IDs to
          // filter and names to CSV list to display as summary's "Sources" value
          if (reportArgs.containsKey("correlatedApps")){
             List apps = reportArgs.getList("correlatedApps");
```

```
            if (apps != null){
               ops.addFilter(Filter.in("links.application.id", apps));
           List appNames = ObjectUtil.convertIdsToNames(context, Application.class, 
apps);
               sources = Util.listToCsv(appNames);
            }
         }

         List fields = new ArrayList();
         fields.add("correlated");
         fields.add("count(*)");

         int correlated = 0;
         int uncorrelated = 0;

         // get counts per Identity with links on the named applications,
         // subdivided by "correlated" flag
         Iterator results = context.search(Identity.class, ops, fields);
         while(results.hasNext()){
            Object[] row = results.next();
            int count = Util.otoi(row[1]);

            // add counts to correlated or uncorrelated totals based on correlated
            // flag
            if ((Boolean)row[0]){
               correlated += count;
            } else {
               uncorrelated += count;
            }
         }
         // calculate percentage of accounts that are correlated
         float percent =  correlated != 0 ? (float)uncorrelated/correlated : 0;
         String percentString = ((int)Math.floor(percent * 100)) +  "%";

         // add values to hashmap; these name/value pairs are displayed in the
         // report summary through the XML's LiveReportSummaryValue elements
         Map map = new HashMap();
         map.put("sources", sources);
```

```
            map.put("correlatedIdentities", correlated);

            map.put("uncorrelatedIdentities", uncorrelated);

            map.put("totalIdentities", correlated + uncorrelated);

            map.put("percentCorrelated", percentString);


            return map;


        </Source>

    </DataSourceScript>
```

A datasource rule would be specified as a nested element (<DataSourceRule>) that contains a rule reference.

```
<DataSourceRule>

<Reference class="sailpoint.object.Rule" id="4028460238ed9b8e0138ed9beff9090f"
name="UncorrelatedAcct Report Summary Rule"/>

</DataSourceRule>
```

The datasource script or rule returns a hashMap of values that are used to populate the corresponding LiveReportSummaryValue elements (based on their name attributes) in the ReportSummary's Values list. The LiveReportSummaryValue elements each include a unique name and a label attribute. The label can be specified as a string or a localizable message key and is displayed alongside the value in the Report's summary section.

```
<ReportSummary title="Uncorrelated Account Details">
    <DataSourceScript>
        <Source>
            …
            Map map = new HashMap();
            map.put("sources", sources);
            map.put("correlatedIdentities", correlated);
            map.put("uncorrelatedIdentities", uncorrelated);
            map.put("totalIdentities", correlated + uncorrelated);
            map.put("percentCorrelated", percentString);

            return map;
        </Source>
    </DataSourceScript>
    <Values>
      <LiveReportSummaryValue label="rept_uncorrelated_ids_grid_label_auth_sources"
name="sources"/>
       <LiveReportSummaryValue label="rept_uncorrelated_ids_summary_correlated"
name="correlatedIdentities"/>
       <LiveReportSummaryValue label="rept_uncorrelated_ids_summary_uncorrelated"
name="uncorrelatedIdentities"/>
       <LiveReportSummaryValue label="rept_uncorrelated_ids_summary_total_ids"
name="totalIdentities"/>
       <LiveReportSummaryValue label="rept_uncorrelated_ids_summary_percent"
name="percentCorrelated"/>
    </Values>
</ReportSummary>
```

Chart: Report Graph

The Chart element defines the graph that is displayed in the Summary section of the report. The chart can be represented as a pie chart or as a column or line graph. The chart data is based on the dataset for the report detail grid (the DataSource element) unless a DataSourceRule or Script is specified for the chart. Commonly, the chart represents the report body's data, grouped and counted by groupings (i.e. “value” is a count, grouped by the category and/or series).

Attributes available to define the chart are listed in this table:

Attribute	Usage
title	Name to display above the chart
type	Identifies the type of chart to display (pie, column, or line)
category	Defines the X axis in line or column graphs; defines the separate sections of a pie graph
value	Defines the Y axis in line or column graphs; defines the portion (fraction) of the pie that belongs to each section in a pie graph; often a count
series	Defines the separate columns or lines on line or column graphs; ignored for pie charts
groupBy	String value of column name (or CSV list of column names) to group data by for graphing counts
sortBy	List of sort columns for data; seldom used since groupBy can specify multiple fields in a CSV list
limit	Limits the number of records examined for the graph; seldom used, unless a similar limit was imposed on the report detail data, because the graph should generally represent all of the data in the report detail
script	Used to define a datasource for the chart other than the report detail data; script contains a <Source> element with beanshell content
dataSourceRule	Used to define a datasource for the chart other than the report details data; contains a reference to a rule where the beanshell has been encapsulated
nullSeries	Label to display if the series value (x-axis) is null (for example, a null certification action status means "Open" so that should be the label for the group of certifications whose action.status is null)
nullCategory	Label to display for data group when the category value is null

In most cases, the chart is created by selecting the value, category, and series fields from the object queried by the report detail datasource using the exact same filter criteria used by that datasource. The table below describes how the chart data is retrieved by default (when a dataSourceRule or Script are not specified for the chart).

Report Detail DataSource Type	Default Chart Query
Filter	DataSource's objectType object is queried using the queryOptions built from the DataSource Query, QueryScript, and QueryParameters elements

Report Detail DataSource Type	Default Chart Query
Java	Requires the DataSource class to have implemented the getBaseQueryOptions method; this method should return the QueryOptions used in the report detail query; also requires that the objectType is specified on the DataSource; retrieves the data for the chart from the objectType object using the QueryOptions returned by getBaseQueryOptions()
HQL	Uses the same query string employed by the report detail query, which specifies both the From and the Where clauses, to retrieve the chart's data

Standard Chart Examples

This example pie chart from the Uncorrelated Accounts Report examines the uncorrelated account data pulled from Links, groups it by application.id and counts the number of uncorrelated accounts on each application. The graph represents the number of uncorrelated accounts from each application (in this case, one uncorrelated account was found per application).

```
<Chart category="application.name" groupBy="application.id"
title="rept_uncorrelated_ids_chart_title" type="pie" value="count(*)"/>
```

The Manager Access Review Report contains an example of a Column graph that shows the count of certification items that are open, approved, or revoked (reflected in the action.status attribute), separated by roles and additional entitlements if applicable (reflected in the type attribute).

```
<Chart category="type" groupBy="action.status,type" nullSeries="cert_action_open"
series="action.status" title="rept_cert_chart_title" type="column"
value="count(*)"/>
```

Chart Script and DataSourceRule

The script and dataSourceRule elements can provide the report writer more flexibility in creating charts based on any data. However, these are generally used only in rare cases in custom reports. The standard reports do not use a Script or DataSourceRule for their charts.

When either of these is used, the beanshell within them is provided the following parameters:

DataSourceRule or Script Parameters	Contents/Purpose
context	A SailPoint Context for executing the search
args	The TaskDefinition Arguments map
report	The entire LiveReport report definition
baseHql	The from and where clause used in the report detail search if the report DataSource was an HQL datasource; null if DataSource type was not HQL
baseQueryOptions	The QueryOptions (specifying the “where” clause criteria) used in the report detail search if the report DataSource was a Filter datasource; null if Datasource type was not Filter

The code must return a list of maps (List<Map<String, Object>>) with each map representing the value, category, and series for each component of the chart. If there is no series, the “series” should be recorded as empty string ("") rather than null.

For example: {[("value","23"), ("category","ADAM"),("series","")],

[("value","5"), ("category","Financials"),("series","")],

[("value","12"), ("category","PeopleSoft"),("series","")]}

The rule or script can add onto the existing criteria from the report detail's DataSource by modifying the baseHql or baseQueryOptions or it can build the chart data with completely independent criteria. The beanshell is responsible for specifying the desired columns and search criteria, executing the database search to retrieve the data, formatting the data into the list of maps, and returning that list.

Report Forms

The layouts and contents of report-specific form pages are specified within a Form object, referenced by the report XML in a <ReportForm> element, nested within the <LiveReport> report definition. The Form object must be created and imported into IdentityIQ separately and is referenced by name.

Each <Section> defines a separate page on the Edit Report window. The page name, shown in the Sections list and at the top of the form, is specified as the Section's label attribute.

```
   <Form name="Uncorrelated Accounts Report Custom Fields">
      <Section label="Uncorrelated Accounts Parameters" name="customProperties">
<Field displayName="report_input_correlated_apps" filterString="logical==false
&amp;&amp; authoritative==false"
helpKey="rept_input_uncorrelated_ident_report_correlated_apps"
name="correlatedApps" type="Application" value="ref:correlatedApps"/>
      </Section>
    </Form>
```

Reports with large numbers of available parameters often include multiple report-specific-parameter pages, specified as multiple Sections in the Form XML, to group the parameters by category.

```
 <Form name="Identity Report Options Form Skeleton">
    <Section columns="2" label="rept_priv_access_section_priv_account_attrs"
name="Priviledged Account Attributes">
       <Attributes>
          <Map>
            <entry key="subtitle" value="rept_priv_access_section_instructions"/>
          </Map>
       </Attributes>
     </Section>
   <Section columns="2" label="rept_priv_access_section_account_props" name="Account
Properties">
        <Field columnSpan="1" displayName="rept_identity_roles_field_app"
helpKey="rept_identity_roles_helpN_app" multi="true" name="applications"
type="Application" value="ref:applications"/>
    </Section>
    <Section columns="2" label="rept_priv_access_section_identity_props"
name="Identity Properties"/>
    <Section columns="2" label="rept_priv_access_section_identity_extended_props"
name="Identity Extended Properties"/>
</Form>
```

Note: Several of the sections in this example XML do not contain any Field definitions. This is because this report uses an initialization rule to create the form fields for those sections based on system data. See Initialization Script or Rule for more information on these dynamic forms.

These are the attributes that can be specified for the Section element.

Section Attribute	Usage
label	The label for the form; can be a string or a localizable message key
name	Name for the section; must be unique per form; used programmatically but not displayed on the UI Edit Report window
Columns	Used to specify the number of columns in which fields should be displayed; fields are displayed in the order they are listed within the section, with one field added to each column in a repeating pattern; for example, in a 2-column layout, 5 fields would be displayed like this: Field 1 Field 2 Field 3 Field 4 Field 5 This attribute can be omitted for a one-column display.

Fields on each form are specified as nested <Field> elements within each <Section>. Important report-form attributes on the Field element are described below.

```
<Field displayName="report_input_correlated_apps"

filterString="logical==false &amp;&amp; authoritative==false"
helpKey="rept_input_uncorrelated_ident_report_correlated_apps"
name="correlatedApps" type="Application" value="ref:correlatedApps"/>
```

Field Attribute	Usage
displayName	The label for the field. Can be a string or a localizable message key.
helpKey	The tool tip for the field. Can be a string or a localizable message key.
name	Name for the field and must be unique per form.
type	Field type. If this entry is an object, it is automatically created as a suggest, allowing the user to select from the system's existing objects of that type.
filterString	A filter string that restricts the set of objects presented to the user for selection. Only applies to objects that are presented as suggest boxes.
value	A reference to the XML input parameter from which to retrieve the starting / default value for the field. This is how the saved values in customized report instances are populated on the form when those instances are viewed in the Edit Report window. Input parameters to the TaskDefinition XML are specified in its signature, as described in Report Signature: Passing Data from Saved Report Instances below.
postBack	A flag indicating if the form should be submitted when the field value changes.This causes the form to be reloaded and any initialization actions to be performed. This flag is important if an initializationScript or Rule or an ExtendedColumnScript or Rule needs to run to add or remove fields on the form or columns on the report based on this field value.

Field Attribute	Usage

AllowedValuesDefinition	This is specified as a nested element to provide a list of values from which the form user can select. See the User Activity Report for an example (excerpted below).
	
	`<Field columnSpan="1" displayName="label_action" helpKey="rept_input_app_activity_report_action" multi="true" name="action" type="string" value="ref:action">`
	`   <AllowedValuesDefinition>`
	`      <Script>`
	`        <Source>`
	`          import sailpoint.object.*;`
	
	`          List items = new ArrayList();`
	`          for(ApplicationActivity.Action action : ApplicationActivity.Action.values()) {`
	`              List l2 = new ArrayList();`
	`              l2.add(action.toString());`
	`              l2.add(action.getMessageKey());`
	`              items.add(l2);`
	`          }`
	`          return items;`
	`      </Source>`
	`     </Script>`
	`   </AllowedValuesDefinition>`

Custom report forms are presented to users in the Edit Report window, along with the Standard Properties page and the Report Layout page. The Standard Properties and Report Layout pages are standard components of the report architecture and are automatically presented for any standard or custom report that implements the LiveReport architecture, whether or not the report references a custom form for report-specific parameters. The Standard Properties page is always presented first and the Report Layout page is always last; report-specific form pages are inserted between these two on the Edit Report window.

Chapter 19: Reports DataSource Example

The following sample report uses a Java data source. This sample displays Identities' name, display name, and manager status and can be filtered by manager (the manager to whom the Identities report) and application (application(s) on which the Identities have accounts). Both of these filters are multi-selectable.

```
<TaskDefinition name="Sample Report"
executor="sailpoint.reporting.LiveReportExecutor"
  subType="Identity Reports" resultAction="Rename"
  progressMode="Percentage" template="true" type="LiveReport">
   <Description>Sample report</Description>
   <RequiredRights>
       <Reference class="sailpoint.object.SPRight"
         name="FullAccessBusinessRoleMembershipReport"/>
   </RequiredRights>
   <Attributes>
      <Map>
        <entry key="report">
           <value>
             <LiveReport title="Manager Status Report">
               <DataSource type="Java"
                 dataSourceClass="sailpoint.reporting.datasource.SampleDataSource"
                 defaultSort="name">
                  <QueryParameters>
                   <Parameter argument="applications"
                     property="links.application.id"/>
                   <Parameter argument="managers" property="manager.id"/>
                  </QueryParameters>
               </DataSource>
               <Columns>
             <ReportColumnConfig field="name" header="Identity Name" property="name"
sortable="true"/>
                 <ReportColumnConfig field="displayName" header="Display Name"
sortable="true"/>
                 <ReportColumnConfig field="managerStatus" header="Is Manager"
property="managerStatus" sortable="true"/>
               </Columns>
             </LiveReport>
           </value>
        </entry>
      </Map>
   </Attributes>
   <Signature>
       <Inputs>
          <Argument multi="true" name="applications" type="Application"/>
          <Argument multi="true" name="managers" type="Identity"/>
       </Inputs>
   </Signature>
</TaskDefinition>
```

This Java datasource, SampleDataSource.java, builds and runs the query for this report based on the filters the user specifies.

```
/* (c) Copyright 2012 SailPoint Technologies, Inc., All Rights Reserved. */
package sailpoint.reporting.datasource;

import net.sf.jasperreports.engine.JRException;
import net.sf.jasperreports.engine.JRField;
import sailpoint.api.sailpointContext;
import sailpoint.object.Attributes;
import sailpoint.object.Filter;
import sailpoint.object.Identity;
import sailpoint.object.LiveReport;
import sailpoint.object.QueryOptions;
import sailpoint.object.Sort;
import sailpoint.task.Monitor;
import sailpoint.tools.GeneralException;
import sailpoint.tools.Util;

import java.util.Arrays;
import java.util.Iterator;
import java.util.List;

public class SampleDataSource implements JavaDataSource {

    private Monitor monitor;
    private sailpointContext context;
    private QueryOptions baseQueryOptions;
    private Integer startRow;
    private Integer pageSize;

    private Object[] currentRow;
    private Iterator<Object[]> iterator;

    public void initialize(sailpointContext context, LiveReport report,
Attributes<String, Object> arguments, String groupBy, List<Sort> sort) throws
GeneralException {
        this.context = context;

        baseQueryOptions = new QueryOptions();

        if (arguments.containsKey("applications")){
            List<String> applicationIds = arguments.getList("applications");
           baseQueryOptions.add(Filter.in("links.application.id", applicationIds));
        }

        if (arguments.containsKey("managers")){
            List<String> managersIds = arguments.getList("managers");
            baseQueryOptions.add(Filter.in("manager.id", managersIds));
        }

        if (sort != null){
            for(Sort sortItem : sort) {
                baseQueryOptions.addOrdering(sortItem.getField(),
sortItem.isAscending());
            }
        }

        if (groupBy != null)
            baseQueryOptions.setGroupBys(Arrays.asList(groupBy));
    }

    private void prepare() throws GeneralException{
        QueryOptions ops = new QueryOptions(baseQueryOptions);

        if (startRow != null && startRow > 0){
            ops.setFirstRow(startRow);
        }

        if (pageSize != null && pageSize > 0){
            ops.setResultLimit(pageSize);
        }

        iterator = context.search(Identity.class, ops, Arrays.asList("name",
"displayName", "managerStatus"));
    }

    public boolean next() throws JRException {

        if (iterator == null){
            try {
                prepare();
            } catch (GeneralException e) {
                throw new JRException(e);
            }
        }

        if (iterator.hasNext()){
            currentRow = iterator.next();
            return true;
        }

        return false;
    }

    public Object getFieldValue(String field) throws GeneralException {
        if ("name".equals(field)){
            return currentRow[0];
        } else if ("displayName".equals(field)){
            return currentRow[1];
        } else if ("managerStatus".equals(field)){
            return currentRow[2];
        } else {
            throw new GeneralException("Unknown column '"+field+"'");
        }
    }

    public void setLimit(int startRow, int pageSize) {
        this.startRow = startRow;
        this.pageSize = pageSize;
    }

    public int getSizeEstimate() throws GeneralException {
        return context.countObjects(Identity.class, baseQueryOptions);
    }

    public void close() {

    }

    public Object getFieldValue(JRField jrField) throws JRException {
        String name = jrField.getName();
        try {
            return getFieldValue(name);
        } catch (GeneralException e) {
            throw new JRException(e);
        }
    }

    public void setMonitor(Monitor monitor) {
        this.monitor = monitor;
    }

    public QueryOptions getBaseQueryOptions() {
        return baseQueryOptions;
    }

    /**
```

```
     * Unused since this is not an hql report.
     */
    public String getBaseHql() {
        return null;
    }
}
```

Managing Passwords

This section contains the following information:

Chapter 20: Introduction to Password Management

IdentityIQ supports multiple login configurations, including single sign-on, pass-through authentication, and validation against IdentityIQ's internally stored passwords. Pass-through authentication and internal passwords can be managed through the IdentityIQ user interface.

IdentityIQ's internal set of passwords are governed by the IdentityIQ password policy. These internal passwords are always available as a fallback login validation for IdentityIQ, even when other authentication methods are used; either the user or an administrator can reset an internal password through IdentityIQ's change password options.

When pass-through authentication is used, IdentityIQ enables the specification of challenge questions that can enable users to reset their own forgotten passwords, once they authenticate to IdentityIQ by correctly answering those questions. New passwords entered through this forgot password feature are validated against the pass-through authentication application's password policy and are reset on that application directly.

This section is divided into three chapters, each covering key subjects related to Password Management:

- "Application Password Management" on page 281
- "IdentityIQ Password Management" on page 291
- "Application-Specific Password Management Requirements" on page 297

Chapter 21: Application Password Management

IdentityIQ can use the Lifecycle Manager product to manage passwords across many of the applications with which it is associated. It can enforce password policies specified for the applications, which can include requirements for length, complexity, unique history, and mandatory reset.

To manage passwords across application, you must configure both IdentityIQ and the applications on which you are going to manage passwords. Password management is further governed by the capabilities of the connector in use for each application and some applications have specific configurations requirements that go beyond the basic password management requirements.

Enabling Password Management in IdentityIQ

The ability to manage passwords in other applications through IdentityIQ is controlled by the Lifecycle Manager Configuration settings for the installation. To configure Lifecycle Manager, click the gear icon and select **Lifecycle Manager Configuration**.

In Lifecycle manager, request permissions are configurable for Identities in four categories:

- **Self Service**: specifies the types of requests that can be made by an Identity on their own behalf
- **Managers**: specifies the types of requests that can be made for other Identities (and the set of Identities for which those requests can be made) when the requester is a Manager (isManagerflag = True)
- **Help Desk Personnel**: specifies the types of requests that can be made for other Identities (and the set of Identities for which those requests can be made) when the requester has the Help Desk Personnel Capability in IdentityIQ
- **All Users**: Specifies the types of requests that can be made for other Identities (and the set of Identities for which those requests can be made) by any user

The password management function can be turned on for any or all of these categories by selecting the **Manage Passwords** Lifecycle Action under each request-permission category.

Defining Special Characters Available Password Use

IdentityIQ enables you to define the special characters that can be used in passwords throughout your deployment of the product. A default set of special characters are included in the System Configuration object.

The special characters enabled for use in passwords are listed in the passwordSpecialCharacters key. To edit these items:

- Click the **Gear** icon in the navigation menu, go to the **Global Settings ->** IdentityIQ **Configuration -> Password tab -> Password Policy** area, and click **Define Character Type**.
  Or
- Go to the debug page of the IdentityIQ user interface and select object type Configuration from the drop-down menu. Select the SystemConfiguration object and edit the value for the passwordSpecialCharacters entry key. For example:

```
<entry key="passwordSpecialCharacters" value="~!@#$%^*_+-={}\\][:;?,."/>
```

Configuring Applications for Password Management

Password management is further governed by the capabilities of the connector in use for each application. Passwords can be managed through IdentityIQ for any application using a read-write connector that has the PASSWORD feature enabled; this feature is enabled when the features String attribute on the application contain the word “PASSWORD”. The application definition, including its featuresString attribute, for each application is viewable in the XML representation of the Application object (accessible from the debug pages or from the iiq console).

```
<Application connector=”sailpoint.connector.LDAPConnector” created=”1334252935835”
featuresString=”AUTHENTICATE, PROVISIONING, ENABLE, PASSWORD, MANAGER_LOOKUP,
SEARCH, ACCOUNT_ONLY_REQUEST" id="4028833636890f860136a7ac1a6c054f"
modified="1335456303423" name="ADAM Direct" profileClass=”“ type=”ADAM - Direct">
```

Note: Not all read-write connectors have the PASSWORD feature enabled. The Connector Registry entry for each connector includes all the valid features for that connector in its featuresString attribute. Specifying PASSWORD in the featuresString of an application to which the feature does not apply does not successfully enable password management for the application. To view the Connector Registry entries from the debug pages, select Configuration from the Objects list and click List. Then click ConnectorRegistry to view the connector registry XML.

Configuring Password Policies for an Application

Password policies specify the password requirements for an application. These can include minimum and maximum lengths for the password and requirements for its makeup (number of letters, digits, uppercase letters, lowercase letters, special characters). The policies can also restrict password choice based on matches in password history, the password dictionary, the Identity's list of attributes, and the Identity's account attributes.

A separate password policy can be defined for each application in IdentityIQ. In fact, multiple policies can be defined for each application.

Defining a Password Policy

Complete these steps to define an application's password policy:

1. Open the application definition. From the navigation menu, go to **Applications -> Application Definition ->** [select application from list or click **Add New Application** to create a new application]. Then click the **Password Policy** tab.

2. Click **Create New Policy** to create a new password policy, click a policy name in the list to edit an existing policy, or click **Add Existing Policy** to select a predefined password policy from the drop-down list, see “Policy Re-Use” on page 284.
3. Name the policy (required) and provide a brief description. Specify any required password characteristics. Most of these characteristics are self-explanatory; these few are further explained here since they might be unclear:
   - **Password history length:** specifies number of previous passwords in password history to check against for uniqueness (prevents re-use of a password over the specified number of password changes); the current password is included in the count
   - **Validate passwords against the password dictionary**: compares password to an internally-stored, implementation-specific password dictionary, ensuring that the password is not, and does not contain, any word in that dictionary (see Password Dictionary below)
   - **Validate the password against the identities list of attributes**: ensures that values stored as Identity attributes (for example, last name, department, office number, region) are not used as the password
   - **Validate the password against the identity’s account attributes**: prevents values stored on the application account from being used as the password

   Note: The password history, if a Password history length value is specified, it is stored as a <PasswordHistory> element on the <Link> (account representation) within the Identity object. It is stored as a comma separated values list of encrypted passwords. The number of passwords stored is determined by the Password history length value specified. New passwords set for the account cannot match any password in the list.
4. Select an **Identity Filter** if this policy should only apply to certain sets of Identities. The default Identity filter is All, which means the policy applies to all Identities. Other options are:
   - **Match List**: specify Identity Attributes or Application Attributes/Permissions by which Identities can be matched for this policy to apply (for example, Identity Attribute: Department = Accounting)
   - **Filter**: specify a filter (as CompoundFilter XML) that can be used to identify Identities to which this policy applies
   - **Script**: specify a segment of beanshell that selects Identities that should use this policy
   - **Rule**: specify a rule (type: IdentitySelector) that returns a list of Identities to which this policy should apply
   - **Population**: apply this filter to the Identities in an existing IdentityIQ Population

   Note: The first policy defined should be the default policy that applies to all users. This policy serves as the “fallback” policy if none of the more restrictive policy Identity Filters apply to the Identity whose password is being validated. If more than one policy is specified with Identity Filter = All, only the last one created is applied in any Identity password validation. This is further explained in the Password Validation Process section.

Password Dictionary

The password dictionary is a set of words (or character strings) that have been deemed impermissible as passwords or password contents for the specific IdentityIQ installation. It is populated by importing a Dictionary XML object through the iiq console or the Import from File option under System Setup. The XML looks like this, and the prohibited words in the password dictionary are included as <DictionaryTerm> elements:

```
<?xml version='1.0' encoding='UTF-8'?>

<!DOCTYPE sailpoint PUBLIC "sailpoint.dtd" "sailpoint.dtd">
```

```
<sailpoint>
   <ImportAction name='merge'>
     <Dictionary name="PasswordDictionary">
      <Terms>
        <DictionaryTerm value="password"/>
        <DictionaryTerm value="identity"/>
      </Terms>
     </Dictionary>
   </ImportAction>
</sailpoint>
```

Include the <ImportAction name='merge'> element to add new terms to the dictionary without overwriting the existing dictionary entries. Omit this element to overwrite the dictionary with a new set of terms.

If removing terms or replacing the entire dictionary, then delete the dictionary object first using the console or debug pages. The terminator will handle removing both the Dictionary and the dictionaryTerms.

Note: Terms included in this dictionary are prohibited even as any part of a password when password dictionary validation is enforced. For example, if the term “rock” were included in the password dictionary, these passwords would all be prohibited: rocketlauncher, sprocket, Sh@mrock125. Additionally, validation against the password dictionary is case insensitive, so RocKeTTe would also be prohibited in this case.

Policy Re-Use

Previously created policies (for example, ones created for one application but applicable to more than one application) can be added to an application instead of recreating the same policy over and over. For example, if super-user accounts on all applications have the same password requirements, the super-user policy could be created once and copied to all applications.

To copy an existing policy from one application to another:

1. On the **Password Policy** tab for the target application, click **Add Existing Policy.**
2. Select the desired password policy by name. The password policy characteristics are displayed for review.
3. Configure the **Identity Filter** to apply the policy to the appropriate set of Identities for this application. Filters might differ from one application to another (for example, different application attributes or permissions or different Identities attributes can designate a super-user on one application but not on another), so they do not carry over between applications in this policy sharing feature.
4. Click **Save** to save the filter on this application.

The policy now appears in the application's **Password Policies** list with a warning icon. Hovering over this icon displays the message “Be careful editing this policy, it is also used by another application.” Changes made to the requirements in a shared policy affect all applications using that policy. Changes made to the Identity Filter on these shared policies only affect the individual application's use of the policy.

Password Validation Process

In many cases, the password policy for an application applies to all users, so there is only one password policy per application. Sometimes, more than one policy is created for a single application to specify different password requirements for different levels or types of user access. In the password management process, when a user's password is being changed, the policy checker scans all of the policies that apply to the identity and creates one super-policy that covers all of the restrictions for that user.

If no password policy is defined for the application, no password policy is enforced and any password entered for a password change is accepted by IdentityIQ and passed to the application to be set as the account's new password.

Application Change Password Provisioning Policy

Some applications support more than one password type. For example, Lotus Notes has three different types of passwords that need to be managed, a vault password, a file password, and an internet password. IdentityIQ can be used to configure those applications so that all password types can be managed through a change password provisioning policy.

The change password provisioning policy template is loaded when a change password request is created through Lifecycle Manager. This template is only loaded for change password. The other password management requests are not affected.

The Change Password provision policy is configured on the Provisioning Policy tab of the Application Configuration page.

Requesting a Password Change

Password changes, self-service or for others, are requested through the Manage Access QuickLink for Lifecycle Manager. When the request is submitted, it is immediately processed through a workflow, by default, the LCM Manage Passwords workflow.

By default, application password requests (forgot, expired, or change), either self-service or for others, invoke the LCM Manage Passwords workflow. This workflow's default configuration requires no application-owner or manager approvals on a password change. It creates and processes a provisioning plan that contains the requested password changes and then notifies the user by email when the change is complete.

If the change request is for an account whose application is configured with a Change Password provisioning policy, additional information is required before the change occurs. See “Application Change Password Provisioning Policy” on page 285.

Self-Service Requests

When a user wants to, and is authorized to, change their own password on an application, they must complete these steps in IdentityIQ:

1. From the **Manage Access Quicklink**, click **Change Passwords** and select **For Me**.
2. Select the application account or accounts for which the password is being changed.

Note: Hover over the help text icon () by the application name to review its password policy requirements.

3. Enter the **Current Password** for each account being updated. Enter the new password twice: once in **New Password** and once in **Confirm Password**.
   If more than one application's password is being changed at a time and the new passwords should all be identical, select **Synchronize passwords for selected accounts**. Each of the selected accounts is then prompt for the **Current Password** for that account but the New Password and **Confirm Password** boxes are displayed only once at the top of the window and apply to all applications whose passwords are being changed.
4. Click **Submit** at the bottom of the window to submit all password changes.

   Note: If the entered passwords do not match or if the password does not meet the requirements of all of the application's password policies, an error message is displayed on this window and the password values must be re-entered before the requested changes are successfully submitted.
5. A summary of the requested changes is displayed on the next window. Review this summary and click **Submit** (or click **Cancel** or **Make Additional Changes** if the changes noted in the summary do not match the desired changes). Individual request line items can be deleted from this window by clicking the **x** icon on any row. Comments can be added to any of the change records by clicking the icon in the **Add Comments** column. These comments are stored on the IdentityRequest object, which can be accessed later through the **My Work** > **Access Requests** menu option.

   Note: The password reset only occurs if all requested changes can be made successfully. If the password reset fails, an error message is displayed at the top of the page indicating the failure.

Requests for Others

As described in Enabling Password Management in IdentityIQ, the sets of Identities for which a user can make requests, as well as the types of requests available to each user, depend on the Lifecycle Manager Configuration settings that apply to that Identity. The rest of this section assumes that the logged-in user is authorized to make password requests for the Identity needing a password change.

Complete these steps to reset another user's password on an external application through IdentityIQ:

1. From the **Manage Access Quicklink**, click **Change Passwords** and select **For Others**.
2. Select the Identity for whom the password change is required.
3. Specify the password change method:
   - **Set passwords for the selected accounts**: enter new passwords manually on this window
   - **Synchronize passwords for selected accounts**: apply a single manually entered password to all of the selected accounts (rather than entering a separate new password for each selected account)
   - **Generate passwords for the selected accounts**: allow system to generate new passwords

   Note: When passwords are reset for another user, the system automatically sets a flag that tells the external application to require a password reset upon initial login by the user, so whether the password is manually set or generated, the user is prompted to change it when they first sign in to the target application.

   Note: The Generate passwords for the selected accounts option can be turned on or off from the Lifecycle Manager Configuration window, Additional Options tab. Select or clear the Enable password auto-generation when requesting for others box in the Manage Password Options section.

4. Select the application account or accounts for which the password is being changed.
5. Enter the new password twice - once in **New Password** and once in **Confirm Password** - if prompted.
   - If **Generate passwords for the selected accounts** is selected, the system does not prompt for a new password.
   - If **Synchronize passwords for the selected accounts** is selected, the password prompting occurs one time at the top of the window above the accounts list.
   - Otherwise, each selected application account has a set of password prompt boxes.
6. Click **Submit** at the bottom of the window to submit all password changes.

   Note: If the entered passwords do not match or if the password does not meet the requirements of the application's password policy, an error message is displayed on this window and the password values must be reentered before the requested changes can be successfully be submitted.
7. A summary of the requested changes is displayed on the next window. If the password is a generated password, the password is displayed in the **Password** column. If it was manually entered, it is represented with ***** in that column. Review this summary and click **Submit** (or click **Cancel** or **Make Additional Changes** if the changes noted in the summary do not match the desired changes). Individual line items can be deleted from this window by clicking the icon on any row. Comments can be added to any of the change records by clicking the icon in the **Add Comments** column. These comments are stored on the IdentityRequest object, which can be accessed later through the access request pages.

   Note: The password reset only occurs if all requested changes can be made successfully. If the password reset fails, an error message is displayed at the top of the page indicating the failure.

LCM Manage Passwords Workflow

By default, application password requests (forgot, expired, or change), either self-service or for others, invoke the LCM Manage Passwords workflow. This workflow's default configuration requires no application-owner or manager approvals on a password change. It creates and processes a provisioning plan that contains the requested password changes and then notifies the user by email when the change is complete.

If the change request is for an account whose application is configured with a Change Password provisioning policy, additional information is required before the change occurs.

The default email template for password change notification sends a summary of the change request. This includes the requester, some representation of the new password, and any comments entered on the request (from the **Summary of Requests** window). If the password was system generated, that password is included in the email body. If it was a manually entered password, it is displayed in the email body as ******; in the case of request-for-others password resets, the new password value must be verbally, or otherwise, communicated to the user by the person who made the change.

To direct IdentityIQ to use a different, custom workflow for password management, create a workflow of type LCMProvisioning and select it as the Manage Passwords business process on the **Lifecycle Manager Configuration** window's **Business Processes** tab.

Passwords on New Account Requests

New account requests often contain password values. If you want to use default account-creation passwords that are different from the standard password policy for that application, IdentityIQ uses a configuration setting to govern the enforcement of password policies on account creation.

To enforce password policies on account creation, complete these steps:

- On the **Lifecycle Manager Configuration** page located under the gear icon menu, **Additional Options** tab, select the Check Password Policy rule as the **Password Validation Rule**. Check Password Policy is a rule that is supplied with Lifecycle Manager that validates the password field on an application's provisioning policy against the application's password policy. To write a custom rule, click the button to the right of that box.
- Define a Create provisioning policy for the application that includes a **password** field. This field name must end with `password`, must be of type **Secret**, and must not have its own validation rule specified for the Password Validation Rule to be applied. The connector maps this password provisioning policy field to the application's password field as the account is created.

When the provisioning policy form is presented for completion, by default to the application owner, the value entered in the **Password** field on the form is validated against the application's provisioning policy.

Troubleshooting Password Management with Provisioning Plan Debugging

Password changes are managed as provisioning activities, creating a provisioning plan that reflects the password change as an account modification request. Problems encountered with password management during the early set-up phases can be more easily diagnosed by turning on logging of the provisioning plan for individual applications as a debugging tool. The provisioning plan shows the actions IdentityIQ intends to perform on the account.

To turn on logging, add this XML block to the desired application's XML through the IdentityIQ Debug pages.

```
<!-- Inserting a provisioning configuration to support dumping the
     provisioning plan out to the log file during every execution. -->
<!-- The deleteToDisable flag prevents account deletion activities, changing them
     to disable account requests instead of delete  -->
<ProvisioningConfig deleteToDisable="true">
  <PlanInitializerScript>
    <Source>
       System.out.println("DEBUG: ProvisioningPlan: \n" + plan.toXml());
    </Source>
  </PlanInitializerScript>
</ProvisioningConfig>
```

This writes the provisioning plan to standard out (appearing as shown below). Note that the password is written in the provisioning plan in plain text, so this ProvisioningConfig should not be left in the Application XML in a production environment.

```
DEBUG: ProvisioningPlan:

<?xml version='1.0' encoding='UTF-8'?>

<!DOCTYPE ProvisioningPlan PUBLIC "sailpoint.dtd" "sailpoint.dtd">

<ProvisioningPlan targetIntegration="ADAM">
```

```
  <AccountRequest application="ADAM"
nativeIdentity="CN=Adam.Kennedy,DC=sailpoint,DC=com" op="Modify">
    <AttributeRequest name="password" op="Set" value="test123">
      <Attributes>
        <Map>
          <entry key="preExpire">
            <value>
              <Boolean>true</Boolean>
            </value>
          </entry>
        </Map>
      </Attributes>
    </AttributeRequest>
  </AccountRequest>
  <Attributes>
    <Map>
      <entry key="identityRequestId" value="0000000028"/>
      <entry key="requester" value="admin"/>
      <entry key="source" value="LCM"/>
    </Map>
  </Attributes>
  <Requesters>
    <Reference class="sailpoint.object.Identity"
id="2c901c1e34aa96a70134aa96e40200ba" name="admin"/>
  </Requesters>
</ProvisioningPlan>
```

Chapter 22: IdentityIQ Password Management

IdentityIQ supports multiple login configurations, including single sign-on, pass-through authentication, and validation against IdentityIQ's internally stored passwords. Pass-through authentication and internal passwords can be managed through the IdentityIQ user interface.

IdentityIQ's internal set of passwords are governed by the IdentityIQ password policy. These internal passwords are always available as a fallback login validation for IdentityIQ, even when other authentication methods are used. A user or an administrator can reset an internal password through IdentityIQ's change password options.

When pass-through authentication is used, IdentityIQ enables the specification of challenge questions that can enable users to reset their own forgotten passwords, once they authenticate to IdentityIQ by correctly answering those questions. New passwords entered through this forgot password feature are validated against the pass-through authentication application's password policy and are reset on that application directly.

For information, refer to the *SailPoint IdentityIQ System Configuration Guide*.

IdentityIQ Password Configuration

IdentityIQ supports one-way hashing for following identity secrets:

- IdentityIQ password
- IdentityIQ password history
- IdentityIQ security question answers
- Application password history for external applications, such as Active Directory.

Note: Hashing support for application password history is enabled even if an application does not have a password policy.

To enable one-way hashing of secret values, click the **Gear** icon and select **Global Settings -> IdentityIQ Configuration -> Passwords** tab **-> Configuration**.

IdentityIQ Password Policy

The password policy for the IdentityIQ internally stored passwords is set in the System Setup configuration pages. Click the **Gear** icon and select **Global Settings ->** IdentityIQ **Configuration -> Passwords** tab **-> Password Policy**.

Most of the setting options are the same as the password policy options for application passwords. Unique settings available only for the IdentityIQ password policy are:

- **Define Character Types**: used to define allowable character types: Digits, Uppercase Characters, Lowercase or Non-English Characters, Special Characters. All characters are allowed if these fields are empty.
- **Days until expiration for manually set passwords**: used when a user resets their own password through the Edit Preferences window. This option sets the password expiration date by adding the specified number of days to the current date. The user is required to reset their password the first time they log into IdentityIQ on or after that expiration date.
- **Days until expiration for generated passwords**: used when an administrator resets a user's password through the Identity Cube's Attributes page. This option sets the password expiration date by adding the specified number of days to the current date. The user is required to reset their password the first time they log into IdentityIQ on or after that expiration date.
- **Minimum Hours between password changes**: specifies the amount of time (in hours) that must elapse before a user can reset their own IdentityIQ login password after they have reset it once. This does not prevent an administrator from resetting the user's password and does not prevent the user from resetting the password again immediately after it was reset by an administrator.
- **Require users to enter their current password when setting a new password**: enables a user to change their IdentityIQ password only if they enter the correct current password for the account.

Additionally, the following password policy options might not be immediately clear to a user, so they are described more fully here:

- **Password history length**: specifies number of previous passwords in password history to check against for uniqueness (prevents re-use of a password over the specified number of password changes)
- **Validate passwords against the password dictionary**: validates new IdentityIQ passwords against the password dictionary (see “Password Dictionary” on page 283)
- **Validate password against the identity’s list of attributes**: ensures that values stored as Identity attributes (last name, department, office number, region, etc.) are not used as the password

The **Validate passwords against the Identity's account attributes** option found on the application password policies does not apply to the IdentityIQ password policy. Those attributes are specific to each application and present a security risk when used in the login credentials for that specific application, but they do not pose the same risk for the IdentityIQ login.

Note: The password history, if a Password history length value is specified, is stored as a <PasswordHistory> element on the Identity object. It is stored as a comma separated values list of encrypted passwords. The number of passwords stored is determined by the value set for the Password history length. IdentityIQ prevents the setting of a new IdentityIQ password for the user that matches any password in the list.

Defining Special Characters for Password Use

IdentityIQ enables you to define the special characters that can be used in passwords throughout your deployment of the product. A default set of special characters are included in the System Configuration object To edit these special characters, go to the **Gear** icon and select **Global Settings ->** IdentityIQ **Configuration -> Passwords** tab and click the **Define Character Types** button. Alternatively, you can go to the debug page of the IdentityIQ user interface. The special characters enabled for use in passwords are listed in the passwordSpecialCharaters key.

Resetting IdentityIQ Internal Passwords

Each user's internally-stored password in IdentityIQ can be updated by that user on the Edit Preferences window. A user with rights to edit Identities' passwords (Password Administrator, Identity Administrator, etc.) can change passwords for other users as well through the Identity Cube.

Note: Passwords set through these options are the internally stored passwords for IdentityIQ. They are used as the primary authentication resource when the default login configuration is used. If pass-through authentication is enabled, the internal password (if one exists) is used to authenticate a user to IdentityIQ if authentication against the pass-through authentication resource fails. This password reset is not pushed out to any external resource.

Self-Service Password Reset

To change your own IdentityIQ password:

1. From the navigation menu bar, click the user name and select **Preferences**.
2. Click the **Password** tab to display the section in which the new password can be entered.
3. If the IdentityIQ password policy requires that the current password be entered, the **Current Password** box appears, and that value must be entered for the password change to be allowed.
4. Enter the new password twice, once in **New Password** and once in **Confirm New Password**. The password must meet the requirements of the IdentityIQ password policy.
5. Click **Save** at the bottom of the window to save the password changes.

Password Resets for Others

Note: To use this feature, you must have authority to reset passwords for other users.

To change an IdentityIQ password for another user:

1. From the navigation menu bar, click **Identities -> Identity Warehouse ->** [select Identity name]. Then click **Change Password** to display the password reset fields.
2. Enter the new password twice (once in **Password** and once in **Confirm Password**). The password must meet the requirements of the IdentityIQ password policy.
3. If this is a temporary password that the user should be prompted to reset, select **Require the user to change their password the next time that they log in**.
4. Click **Save** to save the password change. Password changes for others do not require the user to enter the current password even if that requirement exists for self-service password changes.

Password Expiration Resets

When a password expiration date is set for the IdentityIQ password, the system forces the user to change their password the first time they try to sign in, on or after the specified date.

First the user is informed that the password has expired. Click **Close** to acknowledge and dismiss this message.

Then the user is prompted to enter a new IdentityIQ password. Enter the new password twice (in **New Password** and **Confirm Password**) and click **Change**.

Password Management with Pass-Through Authentication

Note: **This feature is available when pass-through authentication is in use and can only be used to reset the password for a pass-through-authentication application.**

When IdentityIQ is configured for pass-through authentication, the Forgot Password option can be turned on to enable a user to reset their password in the authenticating application. A user can then authenticate to IdentityIQ through security questions when they are unable to remember their password.

To enable this feature, from the Navigation bar, go to the **Gear** icon **-> Global Settings -> Login Configuration -> User Reset** tab and select **Enable Forgot Password**.

This feature causes the **Forgot Password?** link to appear on the IdentityIQ login window. When a user clicks this link, they are prompted to answer one or more security questions that enable IdentityIQ to verify their identity. After a user successfully answers the security questions, the user is prompted for a new password. The pass-through application is then updated with that new password.

Pass-Through Authentication Requirements

Though the setup of pass-through authentication is not the focus of this document, there are a few configurations that are required for Pass-Through Authentication to work. If these configurations are not properly completed, authentication features related to Pass-Through Authentication can be prevented from working.

The **Authentication Search Attributes** field for the application must contain the names of the application account schema attribute(s) that contain the Username entered during sign-on. This field tells IdentityIQ which application fields to search to locate the matching application account. One or more attribute names can be specified in this field.

Defining the Security Questions

To specify the security questions, from the Navigation bar, go to the **Gear** icon **-> Global Settings -> Login Configuration -> User Reset** tab **-> Security Question Configuration -> Questions** area. A default set of security questions is provided. Any of these can be removed from the list by clicking the icon next to the question to be deleted. Custom questions can be defined as needed by clicking the icon next to the last question in the list and entering a new question in the box that appears.

Configuring the Security Question Settings

To configure security questions, from the Navigation bar, go to the **Gear** icon **-> Global Settings -> Login Configuration -> User Reset** tab **-> Security Question Configuration -> Settings** area.

Security Questions

The Security Questions tabs allows users to change security questions and answers, should the user need assistance when the password has been forgotten. The Security Questions tab is only displayed when Forgot Password and Security Question is enabled from the **Login Configuration -> User Reset** page.

Select the desired questions from the three drop-downs and provide the answers in the Answer field.

Click **Save**.

The purpose of each of these settings is described below:

- **Number of questions asked to authenticate an identity** — Specifies the number of correct answers to the security questions the user has to provide to be authenticated by these questions.
- **Number of authentication answers a user must have defined in** IdentityIQ — Specifies the number questions for which the user must provide answers in advance so they can be authenticated using these questions; questions without known answers cannot be used for authentication because there is no “correct” answer to be matched.
- **Prompt users for answers to unanswered security questions upon successful login** — Causes IdentityIQ to check (during login) whether the user has the required number of authentication answers provided already and, if not, prompt the user for those answers.
- **Maximum number of unsuccessful authentication attempts before** IdentityIQ **lockout** — Locks the IdentityIQ account when a user enters invalid authentication answers this number of times.
- **Number of minutes a user will remain locked out due to unsuccessful authentication**: Determines the duration of the lockout before the user can try again to sign in to IdentityIQ. During the lockout period, an administrator with the appropriate system capabilities can unlock the account by clicking **Unlock Identity** on the Identity Cube's **Attributes** tab.

Recording Security Answers

A user can only be authenticated through these questions if the answers are pre-recorded in IdentityIQ. Users can be required to provide these answers or they can choose to provide (or modify) their own answers.

Requiring Security Answers

Users can be forced to provide answers to these questions by selecting **Prompt users for answers to unanswered security questions upon successful login** in the Authentication Questions Settings. This causes the system to check whether each user has the required number of authentication answers recorded during the login process. If too few answers are recorded for a user, the **Answer Authentication Questions** window is display and the user is required to answer these questions before they can gain access to IdentityIQ. The number of questions shown depends on the required number of answers in the Security Question Settings (**Number of authentication answers a user must have defined in** IdentityIQ). The user can select any of the configured questions from the question drop-down lists.

Users who have already provided the required number of answers are not prompted again; this window is bypassed in subsequent logins and they are taken directly to the normal IdentityIQ interface.

Independently Providing or Editing Security Answers

If users are not forced to provide authentication answers, users can choose to provide the answers through the Edit Preferences page. Users can also update their authentication answers on this window, including changing their answers or choosing different questions.

1. From the Navigation menu bar, click the user name and select **Preferences**.
2. Select the **Password** tab.
3. Select the desired questions from the question lists and provide the appropriate answer for each question. Click **Save** to save the changes.

Chapter 23: Application-Specific Password Management Requirements

Some applications have specific configurations requirements that go beyond the basic password management requirements previously discussed in this document.   This section explores some of those application-specific requirements.

Active Directory and ADAM: SSL

Both AD and ADAM require a secure connection (SSL) for any password management activities. IdentityIQ offers two separate read-write connectors for each of these applications.

SSL Configuration for the Direct Connector

Installations using the AD or ADAM Direct connector must generate and install an SSL certificate under AD/ADAM and then build a java key store for IdentityIQ that trusts the AD/ADAM SSL certificate.

These are the basic steps for building that java key store and configuring IdentityIQ to use it.

1. On a Domain Controller, log in as an administrator and open Internet Explorer. Navigate to **Tools** -> **Internet Options** -> **Content** and click **Certificates**.
2. Switch to the **Trusted Root Certificate Authorities Tab** and select the certificate issued by your Active Directory integrated Certificate Server. Click **Export**.
3. Choose **Base-64 encoded X.509(.CER)** as the **Export File Format**.
4. Specify file name for the exported certificate.
5. Finish the export and copy the exported.cer file to the Java client machine.
6. At the client machine run the following command from the jdk bin directory.

   ```
   keytool -import -alias [aliasname] -keystore [keystore filename] -file [fully
   qualified certificate filename]
   ```

   The key store (jks) file is created in the bin directory where the keytool command is found. The name of the file is the name you specified following the -keystore parameter, such as myCaCerts.jks.
7. Create the Application in IdentityIQ using the appropriate direct connector (Active Directory or LDAP - ADAM). Select **Use SSL** and provide all the required values. Save the application (do not click **Test Connection** yet).
8. Assuming that the keystore is created in `/tomcat/apache-tomcat-7.0.47/`, enter the following in `catalina.sh`:
   ```
   -Djavax.net.ssl.trustStore=/tomcat/apache-tomcat-7.0.47/myCaCerts.jks
   -Djavax.net.ssl.trustStorePassword=password
   ```
9. Restart the Tomcat server.

10. Return to the Application Definition in the UI and click **Test Connection** to verify that the SSL connection is properly configured.

Windows Local and Active Directory: IQService Agent

Note: AD and ADAM require a secure connection (SSL) for any password management activities.

The IQService is a native Windows service that enables IdentityIQ to participate in a Windows environment and access information only available through Win32 APIs. You must install and register an IQService before you can provision to Active Directory, aggregate Terminal Services attributes, collect information from the Windows Event Logs, or load local Windows users or groups through the Direct connectors. This includes provisioning of password changes.

IQService can be installed on an independent Windows computer or on a Windows machine that is a member of a domain. It listens for connections from an IdentityIQ instance and can be used to do one of several things, including:

- Aggregate access to the file shares on the server
- Aggregate local user and group definitions from the independent Windows machine
- Aggregate users and groups from the Active Directory or ADAM domain of which the machine is a member
- Change the passwords for a user who has rights to the independent Windows machine or the domain

The application definition for the Active Directory or Windows Local application must then be configured with the host and port where IQService is installed and listening.

Windows Desktop Password Reset Utility

Since a user would normally have to successfully log into their computer before accessing IdentityIQ (or any other application) through a web browser, enabling reset of a Windows Desktop password requires the installation of a utility application called IdentityIQ Lifecycle Manager Desktop Password Reset. This application adds a link or button to the Windows login screen that can be configured to connect users to IdentityIQ's Forgot Password feature (or any other web-based password management solution) in a restricted browser to change their password; this functionality bypasses the Windows login credential requirement for this specific and limited purpose.

Note: Users can only be authenticated and permitted to change the Windows Desktop password through the IdentityIQ Forgot Password functionality if they have previously configured challenge question answers that can be used for authentication.

This utility is available to any customer who has licensed the Lifecycle Manager product.

When this application is installed, the **Forgot Password?** button, tile, or link appears on the login windows.

If configured to point to the IdentityIQ Forgot Password functionality, the restricted browser window displays the IdentityIQ's challenge question authentication windows.