AUTHORIZED DOCUMENTATION

Architecture Reference Guide

Novell®
Identity Manager Resource Kit

1.2

August 17, 2009

www.novell.com

Legal Notices

Novell, Inc., makes no representations or warranties with respect to the contents or use of this documentation, and specifically disclaims any express or implied warranties of merchantability or fitness for any particular purpose. Further, Novell, Inc., reserves the right to revise this publication and to make changes to its content, at any time, without obligation to notify any person or entity of such revisions or changes.

Further, Novell, Inc., makes no representations or warranties with respect to any software, and specifically disclaims any express or implied warranties of merchantability or fitness for any particular purpose. Further, Novell, Inc., reserves the right to make changes to any and all parts of Novell software, at any time, without any obligation to notify any person or entity of such changes.

Any products or technical information provided under this Agreement may be subject to U.S. export controls and the trade laws of other countries. You agree to comply with all export control regulations and to obtain any required licenses or classification to export, re-export or import deliverables. You agree not to export or re-export to entities on the current U.S. export exclusion lists or to any embargoed or terrorist countries as specified in the U.S. export laws. You agree to not use deliverables for prohibited nuclear, missile, or chemical biological weaponry end uses. See the Novell International Trade Services Web page (http://www.novell.com/info/exports/) for more information on exporting Novell software. Novell assumes no responsibility for your failure to obtain any necessary export approvals.

Novell, Inc., has intellectual property rights relating to technology embodied in the product that is described in this document. In particular, and without limitation, these intellectual property rights may include one or more of the U.S. patents listed on the Novell Legal Patents Web page (http://www.novell.com/company/legal/patents/) and one or more additional patents or pending patent applications in the U.S. and in other countries.

Novell, Inc.
404 Wyman Street, Suite 500
Waltham, MA 02451
U.S.A.
www.novell.com

Online Documentation: To access the latest online documentation for this and other Novell products, see the Novell Documentation Web page (http://www.novell.com/documentation).

Novell Trademarks

For Novell trademarks, see the Novell Trademark and Service Mark list (http://www.novell.com/company/legal/trademarks/tmlist.html).

Third-Party Materials

All third-party trademarks are the property of their respective owners.

Contents

About This Guide

The Resource Kit Architecture Guide is a reference guide. It describes the logic, schema, and business logic that are included in the Resource Kit.

Audience

This guide is intended for consultants and architects that are deploying an Identity Manager solution base on the Resource Kit.

Feedback

We want to hear your comments and suggestions about this manual and the other documentation included with this product. Please use the User Comments feature at the bottom of each page of the online documentation, or go to www.novell.com/documentation/feedback.html and enter your comments there.

Documentation Updates

For the most recent version of the *Resource Kit Architecture Guide*, visit the Novell Compliance Management Platform Documentation Web site (http://www.novell.com/documentation/ncmp10/).

Additional Documentation

For documentation on Identity Manager, see the Identity Manager Documentation Web site (http://www.novell.com/documentation/idm36/index.html).

Documentation Conventions

In Novell documentation, a greater-than symbol (>) is used to separate actions within a step and items in a cross-reference path.

A trademark symbol (®, ™, etc.) denotes a Novell trademark. An asterisk (*) denotes a third-party trademark.

When a single pathname can be written with a backslash for some platforms or a forward slash for other platforms, the pathname is presented with a backslash. Users of platforms that require a forward slash, such as Linux* or UNIX*, should use forward slashes as required by your software.

Overview

1

This guide is a reference guide for the Resource Kit. It contains detailed explaination of the Resource Kit configuration and the Identity Vault configuration. The Resource Kit follows best practices for developing an Identity Manager solution.

The information in this guide is intended to help partners rapidly duplicate the common infrastructure of an Identity Manager solution to do certain key tasks:

- Creating a common data model to allow drivers to work together.
- Implementing a common employee life cycle.
- Using entitlements consistently across the drivers.
    - Using Role-Based entitlements or workflows to grant/revoke the entitlement.

Identity Manager Driver Configuration Development Guidelines

2

Even though each driver is unique and uses different policies, all drivers use the same guidelines to make the driver configuration file consistent. For example, all policies and driver configuration files have the same naming conventions and support the same common data module.

These guidelines are used with all shipping drivers and development of any new drivers. If you create a custom driver, use these same guidelines for development.

2.1 Naming Conventions

Naming conventions make it easier to navigate and understand each driver configuration. The naming conventions are defined for all objects that belong to a driver, global configuration values (GCVs), and for user objects created through the Publisher channel.

2.1.1 Driver Configuration Files

The driver configuration file naming convention is:

```
<base name>[-<type>]-IDM<min. engine version>-V<config version>.xml
```

- **Base name:** The name of the connected system or service the driver provides. For example, Active Directory or Delimited Text.
- **Type:** An additional descriptor for the driver configuration file. If there are multiple configuration files, the type distinguishes among the different files.
- **Minimum Engine Version:** Lists the minimum engine version that the driver can run against. The elements to date are:
  - IDM2_0_0
  - IDM2_0_1
  - IDM2_0_2
  - IDM3_0_0
  - IDM3_0_1

- IDM3_5_0
- IDM3_5_1
- IDM3_6_0

- **Configuration Version:** Specifies the particular driver configuration file version. It is a number that is incremented with each release of a new driver configuration file version.
  - V1
  - V2
  - V11
  - V23

For example:

- `ActiveDirectory-IDM3_6_0-V4.xml`
- `DelimitedText-CSVSample-IDM3_6_0-V2.xml`

2.1.2 Driver Objects

Drivers consist of a variety of objects. The naming conventions for these objects make it much easier to understand the driver configuration file.

- **Driver Set:** There is no naming convention for the driver sets.
- **Driver:** There is no naming convention for the drivers.
- **Channel:** Use Designer and iManager to name the channel objects.
  - **Publisher Channel:** Publisher
  - **Subscriber Channel:** Subscriber
- **Driver Policy Objects:** These are policies that exist underneath a driver or a channel object. These policies are usually consumed only by this driver. A driver can contain many policies; without the naming conventions, it is easy to be confused.

  `<channel>-<policyset>[-<feature name>][-<WhatIsThisPolicyDoing>]`

Table 2-1 *Driver Policy Object Naming Convention*

Policy Set	DirXML Script Policy	Style Sheet Policy	Samples
Subscriber Event Transformation	sub-etp	sub-ets	`sub-etp-VetoAdds`, `sub-ets-ChangeRenameToMove`
Subscriber Matching	sub-mp	sub-ms	
Subscriber Creation	sub-cp	sub-cs	
Subscriber Placement	sub-pp	sub-ps	
Subscriber Command Transformation	sub-ctp	sub-cts	
Publisher Event Transformation	pub-etp	pub-ets	`pub-etp-VetoAdds`, `pub-ets-ChangeRenameToMove`

Policy Set	DirXML Script Policy	Style Sheet Policy	Samples
Publisher Matching	pub-mp	pub-ms	
Publisher Creation	pub-cp	pub-cs	
Publisher Placement	pub-pp	pub-ps	
Publisher Command Transformation	pub-ctp	pub-cts	`pub-ctp-HandleFromMerge`, `pub-cts-PasswordSync`
Input Transformation	itp	its	
Output Transformation	otp	ots	
Schema Mapping	smp	sms	

- **Policy Objects in Policy Libraries:** Policy objects in policy libraries might be consumed by more than one driver in different policy sets and channels. The naming conventions for library policies are adapted from the driver policies.

  ```
  lib-<feature name>-<WhatIsThisPolicyDoing>[-<channel>][-<policyset>]
  ```

  - **Lib:** Static prefix to mark the policy as a library policy. This is important so that you can tell which policies belong to that driver and which policies do not.
  - **Feature Name:** Short name that describes the feature this policy is implementing. Examples might be CredProv for Credential Provisioning or PwdSync for Password Synchronization. The feature name groups multiple policies together.
  - **WhatIsThisPolicyDoing:** A compound word or phrase where the words are joined without spaces and are capitalized within the compound word. This word or phrase is a brief descriptive name for the policy.

  For example:

  ```
  lib-CredProv-ConvertPayload-opt
  lib-CredProv-ProcessPayload-itp
  lib-CredProv-RequiredAttributes-sub-cp
  lib-CredProv-Trigger-sub-ctp
  ```

2.1.3 GCV

The GCV naming conventions address the different types of GCVs, the different purposes of the GCVs, and the scopes of the GCVs.

```
[<purpose/scope>.]<group>.[<subgroup>.]<name>
```

- **Purpose/Scope:** The prefix idv (Identity Value) is used with driver set GCVs. Driver-specific values are drv or driver.
- **Group:** Groups GCVs that belong together. Examples for groups could be communication, notification, logging, or security.
- **Subgroup:** Form subgroups within groups, such as smtp or snmp.
- **Name:** Descriptive name for the GCV.

For example:

```
idv.notification.smtp.ip
idv.notification.smtp.user
idv.notification.smtp.pwd
idv.notification.snmp.ip
idv.dit.data.locations
idv.dit.system.rbs
driver.samba.server
```

2.1.4 User Objects

Naming conventions for user objects define variations that each driver supports when generating names for new objects. Even though each company might have very specific naming conventions, there are common options you can use. All options must guarantee tree-wide uniqueness (not just context-wide) as a best practice. This makes login IDs globally unique so that log files of connected systems that are not hierarchical don't run into implementation problems and log files deliver unambiguous information.

All options are base on the name Charley K Browning.

- "Option 1" on page 14
- "Option 2" on page 14
- "Option 3" on page 15

Option 1

This option uses a name-based object name with a variable length for the name.

Pattern	Examples
First character of the given name + surname	cbrowning
First character of the given name + first character of middle name + surname	ckbrowning
First two character of given name + surname	chbrowning
First three characters of the given name + surname	chabrowning
First character of the given name + surname + up to three digits not padded with leading zeroes	cbrowning1

Option 2

This option uses a name-based object name convention with a fixed length.

Pattern	Examples
First character of the given name + up to seven characters of the surname	cbrownin
First character of given name + first character of the middle name + up to six characters of the surname	ckbrowni
First two characters of the given name + up to six characters of the surname	chbrowni
First three characters of the given name + up to five characters of the surname	chabrown

Pattern	Examples
First character of the given name + up to four characters of the surname + three digits padded with leading zeroes	cbrow001

Option 3

This option uses an attribute value as the object name. This option is effective when the publishing application has login ID ownership or has a unique ID that can be used in the implementation. This option offloads the burden to generate a unique ID from the driver and relies on the application to deliver the unique login ID.

Pattern	Example Attribute	Example
Attribute	CN	cbrownin
		chabrown
		cbrow001
	workforceID	615221
		622534
		7252263

2.2 Development Best Practices

There are numerous best practices that can be listed for proper driver development. This guide lists the ones that make sense to improve driver configuration quality, stability, scalability, and readability.

- Section 2.2.1, “When and How to Use GCVs,” on page 15
- Section 2.2.2, “Supported Operations,” on page 16
- Section 2.2.3, “Policy Development,” on page 17

2.2.1 When and How to Use GCVs

GCVs are global configuration values or constants, not global variables. There is no way to change a GCV value at runtime. The GCVs are globally accessible to the driver and driver set, but not to the tree or network.

Given these facts, GCVs are constants and cannot be changed at runtime, but they can be consumed by all drivers in a driver set or by all policies in a driver. This makes GCVs a very powerful configuration tool. Use the following guidelines when developing with GCVs.

- “Using GCVs to Adapt the Driver Configuration File to Changing Environments” on page 16
- “When Not to Use GCVs” on page 16
- “When to Use Driver Set GCVs Versus Driver GCVs” on page 16

Using GCVs to Adapt the Driver Configuration File to Changing Environments

GCVs help driver configuration files adapt to changing environments by externalizing environment-specific, such as placement contexts, domain names, IP addresses, usernames, dates, and times.

It is a bad practice to configure the driver to prompt for information during import and then add the answer directly into policy code. The better approach is to store the answer in a GCV and make the policies reference the GCV. If the answer to the prompt is wrong or the environment changes, the answer also needs to change. It is much simpler to change a single GCV value than to go through all policies that have the value and change the value in each policy.

By adding the configuration data as a GCV, the GCVs become the controls of the driver.

When Not to Use GCVs

If there are certain configuration values that must never change, they should not be surfaced in a GCV.

All information in a GCV can be easily changed or tuned. To keep certain configuration values and implementation details from being changed, add this information directly into the code. Everything that an administrator should see and be able to change should go into a GCV. Everything that only a developer sees or changes should go directly into the driver policy code.

The only reason to add a static value to a GCV is if it is used in many different places and it does not make sense to add it directly to the code.

When to Use Driver Set GCVs Versus Driver GCVs

GCVs can be defined on a driver or driver set. The GCV location determines its scope and visibility. GCVs defined on the driver set are visible to all drivers and their policies in that driver set. GCVs defined on a driver can only be consumed by policies of the driver.

If a GCV defined on a driver has the same name as a GCV defined on the driver set, the driver GCV takes precedence in all policies that belong to that driver. This allows for easy exception handling:

- Driver Set: idv-new-users = users.staging.data
- All drivers except one: none
- Exceptional driver: idv-new-users = users.inactive.data

All drivers that do not explicitly define the GCV g-driver-new-users would inherit this GCV from the driver set, and policies in those drivers can read the value user.data. The policies in the exceptional driver, which defines a GCV with the same name, read the value users.inactive.data.

2.2.2 Supported Operations

There are certain operations that all administrators should be able to rely on having in every driver configuration file. There are other operations that are optional. This is to avoid problems that might occur if an unexpected operation is invoked, and also to establish a similar driver behavior across all drivers.

The following options should be implemented on each driver. For a more detailed explanation of how the Metadirectory engine handles these requests, see "Synchronizing Objects" in the *Identity Manager 3.6.1 Common Driver Administration Guide*.

- "Migrate From Identity Vault" on page 17
- "Migrate Into the Identity Vault" on page 17
- "Resynchronization" on page 17

Migrate From Identity Vault

Over time, event-based synchronization tends to result in a growing gap between what is expected and what actually happens. This is because policy writers do not consider all possible events and don't implement policies to take all necessary actions.

Administrators want and need the ability to tell a driver to re-evaluate the current data in the Identity Vault against its policies and perform all the necessary tasks so that after the evaluation occurs, the gap between the two systems is closed.

Migrate Into the Identity Vault

Over time, event-based synchronization tends to result in a growing gap between what is expected and what actually happens. This is because policy writers do not consider all possible events and don't implement policies to take all necessary actions.

Administrators want and need the ability to tell a driver to re-evaluate the current data in the application against its policies and perform all the necessary tasks so that, after the evaluation occurs, the gap between the two systems is closed.

Resynchronization

Administrators might click the resynchronize button for the same reasons as they would to a migrate data to or from Identity Vault. However, the Metadirectory engine treats a resynchronization request very differently from migration requests. If "Migrate From Identity Vault" on page 17 and "Migrate Into the Identity Vault" on page 17 are implemented fully, there would be no need for the resynchronization option.

2.2.3 Policy Development

The guidelines in this section are not the only options for policy development, but they are backed by years of field experience and also represent engineering experience and advice.

- "Structuring Rules and Policies" on page 18
- "Using Variables Across Policies" on page 18
- "XSLT Versus DirXML Script" on page 18

Structuring Rules and Policies

New policy writers have trouble deciding whether to use a single rule or to use multiple rules, and whether to use a single policy or to use multiple policies. Use these considerations to help you decide:

- Group actions by task, feature, and requirement. This means that if you define a feature called Password Synchronization, you group all the conditions and actions that make up this feature and put them into a single separate policy or multiple separate policies. Do not have different features use the same policy because that makes it difficult to remove or add individual features.
- Every action of a task, feature, or a requirement that must be completed based on the same conditions and that can be completed in the same policy set should be placed into the same rule.

  NOTE: An exception is that if one action changes the variables of the conditions, this action must have its own rule, but it must reside in its own policy.

- All rules that can be executed in the same policy set should be grouped into the same policy.
- A short-circuit evaluation rule should be the first rule in a policy, if a policy deals with numerous rules that all test on very similar but slightly different conditions, and there is a strong possibility that none of the rules will be activated.

  The evaluation rule should opt out of the current policy when it is obvious that none of the conditions in the rules evaluate to True. An example is a Placement policy based on the location code. The Placement policy might be rather long, consisting of a dozen or even hundreds of rules, each testing for their specific location code. If the location code is a well-know format, such as two characters and no numbers, it makes sense to have the first rule of the policy check these requirements and put a break in the policy if the value of the location does not conform to the format, and none of the following rules would even run.

Using Variables Across Policies

If dynamic information, like an attribute value from a query result, is passed on from policy to policy, local variables cannot be used prior to Identity Manager 3.5. The earlier versions limited the scope of a variable to a single policy. GCVs cannot be used either because they are a constant and the content cannot be changed.

The solution is to use an operation property. An operation property can transport typeless text data from policy to policy and can easily be set and queried. Use the `set operation property` action, the `operation property` noun, or the `if operation property` condition to use operation properties. Avoid issuing the same query multiple times if you can.

XSLT Versus DirXML Script

The general recommendation is to create what is possible with reasonable effort in DirXML® Script. XSLT is a valid way of doing things in Identity Manager, but it is easier on other people to understand what a policy is doing, if it is be implemented in DirXML Script.

One limitation of DirXML Script that forces the use of XLST is the XML document does not conform to XDS as specified in the `nds.dtd`. Typical examples here are input and output transformation policies of some of the generic drivers, such as Delimited Text, SOAP, or JMS.

For more information about the `nds.dtd`, see the *Identity Manager 3.6 DTD Reference*.

2.3 Common Data Model

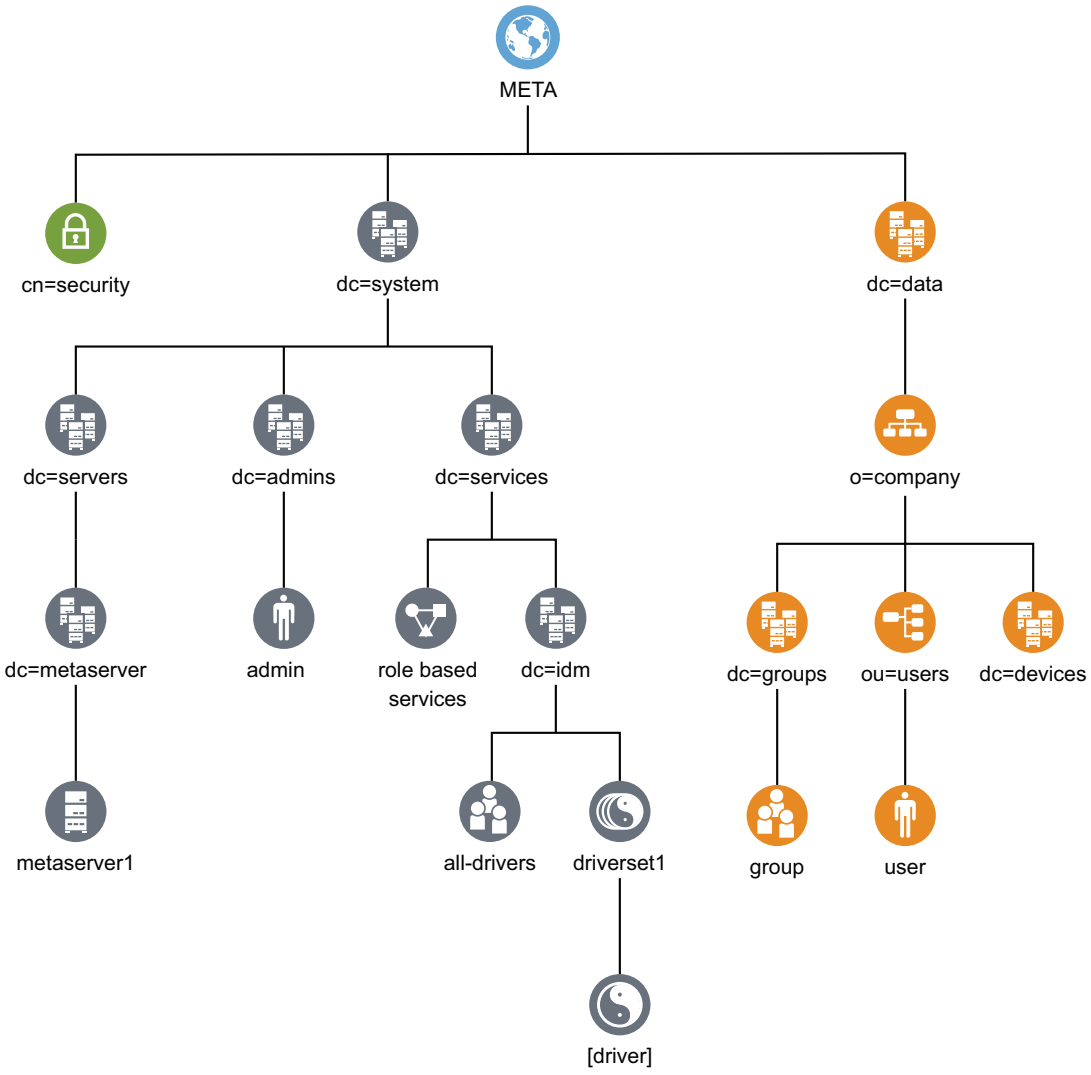
The data model describes the data that the drivers deal with, and the structure and format in which the data is stored in the Identity Vault. Not all drivers support a full core data model. The core data model contains all object classes and attributes that make sense to be available in all Identity Manager installations. Not all objects and values are used in every project, but they should be available for use at any time.

Identity Vault Structure

3

The Identity Vault structure for the Resource Kit is designed to use best practices. The structure is created to use mostly domain (dc) containers instead of traditional organization (o) and organizational unit (ou) containers. Figure 3-1 shows the structure of the Resource Kit Identity Vault.

Figure 3-1 *Identity Vault Structure*



The Resource Kit uses mostly domain containers, so that the sense of relationship that is implied with organizations and organizational units is not there. The recommendation is to use domains if possible, use organizations where it makes sense, and use organizational units only where necessary.

The Resource Kit structure is set up for scalability by having three main components:

- Section 3.1, “Security,” on page 22
- Section 3.2, “System,” on page 22
- Section 3.3, “Data,” on page 23

3.1 Security

The security container is a special container created during the installation of the Identity Vault. It is designated as cn=security instead of dc, o, or ou. This container holds all security objects for the Identity Vault. For example, it contains the certificate authority and password policies.

3.2 System

The system container is a domain. It designated as dc=system. This container holds all of the technical and configuration information for your Identity Vault and for the Identity Manager system. The system container holds three main subcontainers:

- “admins” on page 22
- “servers” on page 22
- “services” on page 22

admins

The admins container holds all of the administrative objects for the Identity Vault and for the User Application. A best practice is to have a separate administrative user object for the User Application and not use the main administrative object for the Identity Vault.

The user padmin.admins.system is the administrative user for the User Application. The main administrative user for the Identity Vault is admin.admins.system.

servers

The server objects have many different objects that are associated with them that must reside in the same container as the server object. As you add more servers into your tree, scrolling through all of those objects can become very cumbersome.

A best practice is to have a separate container for each server object. The name of the container is the name of the server object. All objects associated with the server (volumes, licenses, certificates) are in place and it is much easier to find the objects you need.

This structure is designed for scalability, so if you have 10 or 100 servers, it is easy to find the objects associated with a single server.

services

Any services that are in the Identity Vault are stored in this container. There are multiple subcontainers for the different services. The driver set objects are stored in the idm container, which is a service subcontainer. The Resource Kit only has one driver set, but this structure allows you to scale by adding more driver sets to the idm container. The work order objects and an Organizational Role are also stored in the services subtree container.

3.3 Data

The data container holds groups, users, and devices. This is the data that makes up your system. Resources has three subcontainers: groups, users, and devices. The groups and devices are domains and the users container is an organizational unit.

Business Logic

4

Each company has its own business requirements and policies. Identity Manager allows you to not only automatically create user accounts, but it allows you to add these business requirements into your solution. By doing this, you ensure all business policies are being followed. These business requirements become the business logic that is built into your Identity Manager Solution.

The business logic for the Resource Kit is not contained in each driver. It is implemented through Role-Based Entitlements (RBEs), workflows, service drivers, work orders, and jobs.

4.1 Best Practices

The business logic is kept in multiple places:

- Entitlements Framework
  - Roles
  - Workflows
- Service Agents
- Application Drivers

The order represents the amount of business logic an element is supposed to hold. Most of the business logic and decision making is done in the entitlements framework, which includes roles and workflows. If no automation is possible or wanted or if additional approval is necessary, Workflow can help out. Service Agents can hold business logic that cannot be implemented in roles or workflows. Application Drivers should not contain any business logic at all, but can be used if there is no other option.

The reason to use this order is so the business logic does not need to be duplicated for each connected system. For example, if the business logic resided in policies for the Active Directory* driver, you would have to create the same logic for the Notes driver, Peoplesoft driver, and any additional driver you have in your environment.

By implementing the business logic in the order listed, it saves time, effort, and reduces the chance for errors to occur. If something changes in your business logic, you can make those changes to a single place instead of making the change per connected driver.

4.2 New Hire Business Logic

This section describes the business logic used in the Resource Kit when a new hire occurs. The Resource Kit is kept very generic and flexible so that it can be reused in many different environments quickly. The business logic is contained in multiple drivers, jobs, and work orders.

4.2.1 New Hire without Manager Approval

1. Submit a request for a new employee at the Register New Employee Web page. This Web page is a custom portal page created in the User Application. For more information, see "Importing the Custom Portal Page" in the *Identity Manager Resource Kit 1.2 Installation Guide for the Identity Manager Components*.
2. When this request is issued, a new user account is created in the Identity Vault with a HireDate, FirstWorkingDay, and DirXML-ProcessTable attributes.
3. The next two step occur simultaneously:
   a. The Business Logic driver creates a WorkOrder object with a due date that is equal to the HireDate attribute and a WorkOrder type of Account Enabled.
   b. The State Machine driver starts the ELCP and the Hire process. The ELCP is the parent process.
4. The WorkOrder driver keeps checking the due date on the WorkOrder object, until the due date reaches the current time. The due date can be set in the future.
5. The next two steps occur simultaneously:
   a. When the due date is current, the WorkOrder driver processes any current Account Enabled WorkOrder objects, and creates a WorkToDo object.
   b. When the due date reaches the current time, the Process-Trigger job that is on the State Machine driver runs.
6. The next two steps occur simultaneously:
   a. The Business Logic driver detects the add event for the WorkToDo object and executes the command of Account Enable.
   b. The State Machine driver advances the hire process.

      For more information about the hire process, see Section 5.5, "Hire Process," on page 34.
7. The WorkOrder object sets the attribute of login disabled to false, which enables the user's account.

4.2.2 New Hire with Manager Approval

1. Submit a request for a new employee at the Register New Employee Web page. This Web page is a custom portal page created in the User Application. For more information, see "Importing the Custom Portal Page" in the *Identity Manager Resource Kit 1.2 Installation Guide for the Identity Manager Components*.
2. When this request is issued, a new user account is created in the Identity Vault with a HireDate and a DirXML-ProcessTable attributes.
3. A workflow is created and is sent for approval. The approver could be to a manager, an HR person, or anyone else who must approve hiring a new employee.
4. The new hire is confirmed through the Workflow by the approver.
5. The next two steps occur simultaneously:
   a. The Business Logic driver creates a WorkOrder object with a due date that is equal to the HireDate attribute and a WorkOrder type of Account Enabled.

b. The State Machine driver starts the ELCP process and the Hire process. The ELCP is the parent process.

   For more information about the hire process, see Section 5.5, “Hire Process,” on page 34.

6. The WorkOrder driver keeps checking the due date of the WorkOrder object until the due date reaches the current time. The due date can be set in the future.
7. The next three steps occur simultaneously:
   a. When the due date reaches the current time, a workflow is started and sent to the hiring manager to grant approval for the user’s entitlements. The hiring manager can either approve or deny the entitlements for the user. This also sets the first working day date for the user.
   b. When the due date is current, the WorkOrder driver processes any current Account Enabled WorkOrder objects and creates a WorkToDo object.
   c. When the due date reaches the current time, the Process-Trigger job on the State Machine driver runs.
8. The next two steps occur simultaneously:
   a. The Business Logic driver detects the add event for the WorkToDo object and executes the command of Account Enable.
   b. The State Machine driver advances the hire process.
9. The WorkOrder object sets the attribute of login disabled to false, which enables the user account.

4.3 Termination Business Logic

This section describes the business logic used in the Resource Kit when a termination occurs. The Resource Kit is kept very generic and flexible so that it can be quickly reused in many different environments.

4.3.1 Termination without Manager Approval

1. A termination date is entered into PeopleSoft.
2. This request adds a termination start date to the user object.
3. As soon as the termination start date exists, a work order is created with a termination date. The termination date can be set in the future. The work order keeps checking the termination date until it reaches the current time.
4. As soon as the termination date is the current time, the entitlement for the user is revoked. This is an automated process done by a workflow.
5. As soon as the entitlement is revoked, the process trigger (job), the business logic driver, and the state machine driver work together and compare the termination date versus the last working day. They do not need to be the same day. When they have the termination date and the last working day, the drivers and the job are able to complete the termination process.

   For more information about the termination process, see Section 5.6, “Termination Process,” on page 41.

6. When the termination process reaches the ready to terminate state, another work order sets the employee status to terminated and the login disabled attribute to true.
7. The termination process reaches the terminated state, the employee is terminated, and disabled. All access to resources has been revoked by the entitlements.

4.3.2 Termination with Manager Approval

1. A termination date is entered into PeopleSoft.
2. This request adds a termination start date to the user object.
3. A workflow is created and sent for approval. This could be to a manager, an HR person, or anyone else who must approve terminating an employee.
4. When the termination is confirmed, a work order is created with a termination date. This termination date can be set in the future. The work order keeps checking the termination date until it reaches the current time.
5. As soon as the termination date is the current time, a workflow is started and sent to the manager to grant approval for the user's entitlements to be revoked. This is an automated process done by the workflow. The manager can either approve or deny that the entitlements are revoked.
6. As soon as the termination start date exists, a work order is created with a termination date. The termination date can be set in the future. The work order keeps checking the termination date until it reaches the current time. This also sets the last working day for the employee.
7. As soon as the entitlement is revoked, the process trigger (job), the business logic driver, and the state machine driver work together and compare the termination date versus the last working day. They do not need to be the same day. When they have the termination date and the last working day, the drivers and the job are able to complete the termination process.

   For more information about the termination process, see Section 5.6, "Termination Process," on page 41.
8. When the termination process reaches the ready to terminate state, another work order sets the employee status to terminated and the login disabled attribute to True.
9. The termination process reaches the terminated state, the employee is terminated, and access is disabled. All access to resources has been revoked by the entitlements.

Business Processes

5

Identity Manager automates the process of managing your user's identities in different systems. The business logic implemented in Identity Manager is a reflection of a real business process that was formerly manual, because Identity Manager now has an abstraction layer to help define the business processes before trying to implementing the business processes in policies.

This abstraction layer defines the individual steps and substeps that must be completed during the life cycle of an employee or a resource in a business. There are many different variations that can occur. For example, is the employee a contract worker or a permanent employee? When the employee is terminated, is the account deleted or disabled?

The abstraction layer is implemented through two custom drivers that handle the different scenarios that occur during the life cycle of an employee. The logic is stored in the business logic driver, the state machine driver, jobs, work orders, workflows, and policies associated with these drivers.

5.1 Features of the Abstraction Layer

The abstraction layer has features that allow it to work quickly and flexibly.

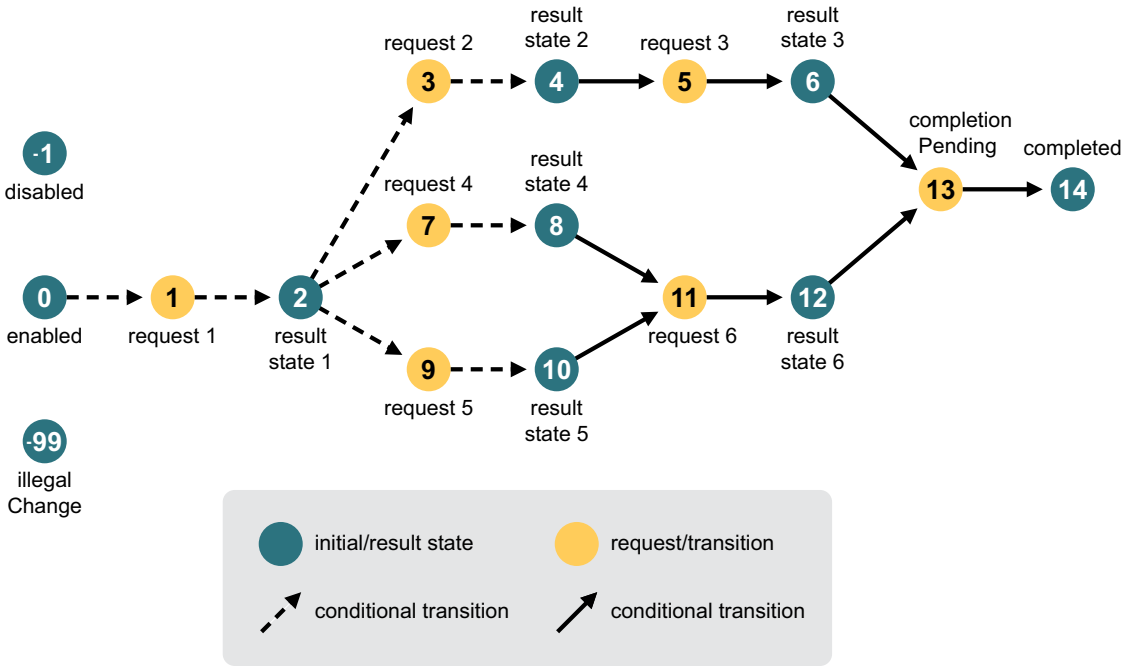
- **Supports any process (extensibility):** Every company is different and so are the processes at the company. The abstraction layer adapts to all of the possible scenarios by allowing for free process definition without any limitations to currently known processes.
- **Allows multiple different processes to run concurrently:** While a hire process is running, a new resource can be requested for the new employee.
- **Allows multiple instances of the same process to run concurrently:** An employee can request several resources at once, and have multiple request resource processes running at the same time.
- **Triggers any process from any data event in the Identity Vault:** Changes to objects in the Identity Vault trigger new processes or advance processes that are already running. This allows for process automation.
- **Triggers any process from any other process:** Any step in a process can trigger another process or advance a current process. This allows for dependencies between processes. For example, a request resource process can start as soon as the hire process reaches a certain step.
- **Identifies each process uniquely:** This allows for tracking of process states, and stops any data corruption from occurring.
- **Audits each process based on the unique identifier:** Each process state can be audited to show the state of each employee or resource by using the unique identifier. If there are problems, the ability to audit each process helps with troubleshooting issues.

- **Reports the state of any process at any time:** For a health check, administrators or auditors can generate reports of the current state of all processes at any time. For example, the health check can report all employees who have a hire or termination process that is not progressing.
- **Allows processes to branch, merge, and loop:** The processes can branch, merge, and loop depending upon defined conditions.
- **Enforces the proper order of steps:** The abstraction layer enforces the proper order of execution for steps. If a violation occurs, it can act on that violation.
- **Separates the process definitions from implementation:** The processes are separate from the implementation in order to rapidly replicate the solution.
- **Runs processes for any object in the Identity Vault:** The process can run for any object, such as a user, person, or resource.

5.2 Definition of a Process

A process is a number of sequential steps that represent requests and results. Figure 5-1is a graphical representation of a process.

Figure 5-1 *A Generic Process*



Every step in the process is represented as an integer. Positive odd numbers represent requests. Positive even numbers are successes, and negative even numbers are failures. Negative odd numbers are reserved for process states and processing errors. Processes can loop, branch into sub-processes, and merge again downstream.

Table 5-1 contains a list of the most common states that occur during a process.

Table 5-1 *Common States*

State	Name	Description
-99	illegalChange	An illegal state change occurred. The legal sequence of steps is disrupted.
-1	disabled	The process is disabled.
0	enabled	The process is enabled but has not started.
*n** -1	completionPending	The next-to-the last state in every process.
n	completed	The last state in every process. The process ends with this step.

The value of *n* is the highest number achieved during the process.

5.3 Schema

In order for the abstraction layer to work, additional schema definitions are added to eDirectory™. The schema extensions add objects and attributes that store the process definitions and process states.

- **DirXML-Processes:** An object class. It is a container object that can be created anywhere in the directory. It is the central repository for the process definition objects. It contains a name and optional description.
- **DirXML-Process:** An object class. It is the process definition object. It stores the definition of the process. Objects of this class are named after the process type used in the process table. It contains a name, an optional description, and the process definition in XML format.
- **DirXML-ProcessDef:** An attribute. It is the process definition in XML format. (Not yet implemented.)
- **DirXML-ProcessesAux:** An auxiliary class. It stores attributes that contain the process table and history. Because it is an auxiliary class, processes can be defined for objects of any class.
- **DirXML-ProcessTable:** An attribute. It contains a list of the currently running processes. After a process is finished, it is removed from this table.

  Each process is represented as a delimited string and the attribute is a substring indexed to make queries efficient. The delimiter used to separate the fields inside a process is the pound sign #. See Table 5-2 for a list of the defined fields.
- **DirXML-ProcessHistory:** An attribute. It contains all of the previous states of a running process. After the process is finished, all entries in the history pertaining to that process are removed. DirXML-ProcessHistory entries follow the same format as the entries in the DirXML-ProcessTable attribute. See Table 5-2 for a list of the defined fields.

Table 5-2 *DirXML-ProcessTable and DirXML-ProcessHistory Fields.*

Field	Format	Description
type	String	Identifies the process through its name, such as hire or termination. The value in this field must match the name of the DirXML-Process object that defines the process.

Field	Format	Description
state key	Signed Integer	Tracks the current state of the process.
pid	String	A unique process identifier.
time	YYYYMMDDhhmmss	The last time and date the process was updated.
state name	String	A human-readable string describing the state. This is closely linked to the state key. It is recommended that camel-case is used and no spaces or special characters are added.
params	String	Parameter necessary to perform the requested step.
msg	String	An optional result status message.

The following example shows how the DirXML-ProcessTable attribute and the DirXML-ProcessHistory attribute are used. This information is taken from an LDIF export of a user object.

```
DirXML-ProcessTable: elcp#10#PID0000124950#20080208013104#hired##
DirXML-ProcessHistory: elcp#0#PID0000124950#20080208010831#enabled##
DirXML-ProcessHistory: hire#0#PID0000124951#20080208010832#enabled##
DirXML-ProcessHistory:
hire#39#PID0000124951#20080208010832#noConfirmationRequired##
DirXML-ProcessHistory: hire#40#PID0000124951#20080208010832#readyForHire##
DirXML-ProcessHistory:
hire#49#PID0000124951#20080208010832#directHirePending##
DirXML-ProcessHistory:
hire#50#PID0000124951#20080208013057#directHireComplete##
DirXML-ProcessHistory:
hire#99#PID0000124951#20080208013058#completionPending##
DirXML-ProcessHistory: elcp#9#PID0000124950#20080208010831#startHire##
DirXML-ProcessHistory: hire#100#PID0000124951#20080208013101#completed##
```

5.4 Employee Life Cycle Process

This section explains how the employee life cycle process (ELCP) works. It manages all HR-driven events that occur during the life cycle of the employee's account, including the hire and termination processes.

Every step in the ELCP process is represented as an integer. Positive odd numbers represent requests. Positive even numbers are successes, and negative even numbers are failures. Negative odd numbers are reserved for process states and processing errors. Processes can loop, branch into sub-processes, and merge again downstream.

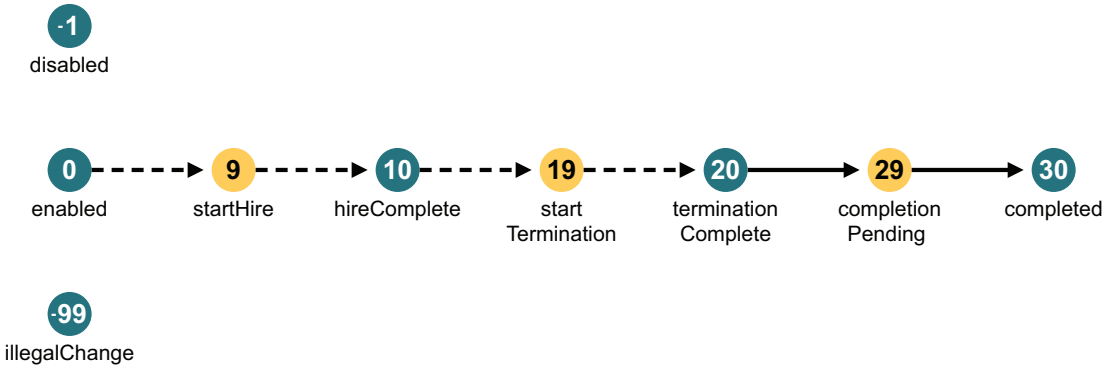
The ELCP starts the hire process when the startHire request is issued. The hire process handles all events that occur during the hiring. The ELCP measures the success of the startHire request and sets the hireComplete state when all requirements are met.

Only a successfully hired employee can be terminated. The ELCP controls when the termination process starts and records the status of the termination process in the terminationComplete event.

The ELCP process currently controls only the hire process and the termination process. In the future, other HR events like maternity leave, promotion, relocation, or demotion can be handled by the ELCP process.

Figure 5-2 is a graphical representation of the ELCP.

Figure 5-2 *Employee Life Cycle Process (ELCP)*



5.4.1 Employee Life Cycle Process States

This section contains a descriptions of each state that can occur in the ELCP. The descriptions refer to global configuration values (GCVs) that are set on the drivers.

enabled (0)

- **Description:** ELCP is enabled.
- **Conditions:** The required conditions for the ELCP process to reach the current state.
  - The object is a user.
  - The object resides in the container defined in the GCV *Operational Settings > User Container*.
  - No other ELCP process is currently running for the object.
  - PID != -1
- **Actions:** The business logic driver verifies the account is disabled (Login Disabled=True) and marked inactive (employeeStatus=inactive).

startHire (9)

- **Description:** Starts the hire process.
- **Conditions:** The required condition for the ELCP process to reach the current state.
  - The current ELCP state is enabled (0).

hired (10)

- **Description:** The hire process is completed.
- **Conditions:** The required conditions for the ELCP process to reach the current state.
  - The current ELCP state is startHire (9).
  - The current hire process state is completed.

startTermination (19)

- **Description:** Starts the termination process.
- **Conditions:** The required conditions for the ELCP process to reach the current state.
  - The current ELCP state is hired (10).
  - The termination date requires a value.

terminated (20)

- **Description:** The termination process is completed.
- **Conditions:** The required conditions for the ELCP process to reach the current state.
  - The current ELCP state is startTermination (19).
  - The current termination process state is completed.

completionPending (29)

- **Description:** ELCP is finishing.
- **Conditions:** The default behavior is that this step is a manual process. This state is set through a GCV. By default, the ELCP does not auto-complete.
  - The current ELCP state is terminated (20).

completed (30)

- **Description:** ELCP is completed.
- **Conditions:** The required condition for the ELCP process to reach the current state.
  - The current ELCP state is completionPending (29).

5.5 Hire Process

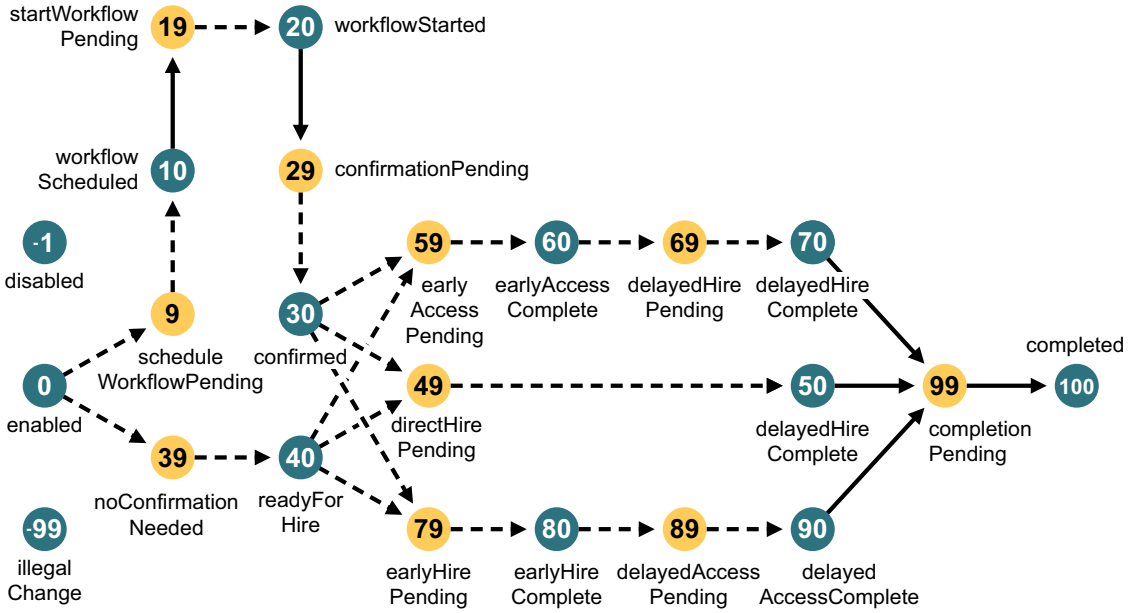
The section describes how the hire process works. Every step in the hire process is represented as an integer. Positive odd numbers represent requests. Positive even numbers are successes, and negative even numbers are failures. Negative odd numbers are reserved for process states and processing errors. Processes can loop, branch off into sub-processes, and merge again downstream.

- Section 5.5.1, “Hire Process Conditions,” on page 35
- Section 5.5.2, “Hire Process States,” on page 35

5.5.1 Hire Process Conditions

Figure 5-3 is a graphical representation of the hire process with all of the variable conditions included.

Figure 5-3 *Hire Process*



5.5.2 Hire Process States

This section contains a description of each state of the hire process. The hire process refers to states that occur in the employee life cycle process (ELCP) as well as to global configuration values (GCVs) that are set on the drivers. For more information on the ELCP, see Section 5.4, "Employee Life Cycle Process," on page 32.

enabled (0)

- **Description:** The hire process is enabled.
- **Conditions:** The required conditions for the hire process to reach the current state:
  - No hire process is currently running for the object.
  - The ELCP state is startHire.
  - The object has a hire date (DirXML-HireDateStr).
  - PID != -1

scheduleWorkflowPending (9)

- **Description:** A transitional state to the workflowScheduled (10) state in the hire process.
- **Conditions:** The required conditions for the hire process to reach the current state.
  - The current state is enabled (0).
  - ELCP is at startHire.
  - The *Manager confirms new employee via workflow* GCV is set to True.
- **Actions:** The expected actions from the components of the solution.
  - The business logic driver generates a random password for new employee if the *Generate random passwords for new employees* GCV is set to True.
  - The business logic driver schedules confirmation workflows by using work orders.
  - The work order driver generates work to do objects when the scheduled date becomes current.

workflowScheduled (10)

- **Description:** The workflow is scheduled to run *n* number of days before the hire date. The value *n* is set through the *Days before hire date to start confirmation* GCV on the driver set.
- **Conditions:** The required condition for the hire process to reach the current state.
  - A valid work order is created
- **Actions:** The expected action from the components of the solution.
  - The business logic driver starts the confirmation workflow.

startWorkflowPending (19)

- **Description:** A transitional state prior to the workflowStarted (20) state.

- **Conditions:** The required condition for the hire process to reach the current state.
  - The current state is workflowScheduled (10).

workflowStarted (20)

- **Description:** The workflow is started and running.
- **Conditions:** The required condition for the hire process to reach the current state.
  - The work order is deleted.

confirmationPending (29)

- **Description:** A transitional state to the confirmed (30) state in the hire process.
- **Conditions:** The required condition for the hire process to reach the current state.
  - The current state is workflowStarted (20).
- **Actions:** The expected actions from the components of the solution.
  - The manager claims and approves or declines the confirmation workflow.
  - If the workflow is approved, the User Application driver grants the confirmed entitlement.
  - If the workflow is declined, the User Application driver the confirmed entitlement.

confirmed (30)

- **Description:** The confirmation workflow is approved. This state represents a hiring manager confirming that a new hire is staring work and agrees to a hire date or a first working day.
- **Conditions:** The required conditions for the hire process to reach the current state.
  - The current state is confirmationPending (29).
  - The entitlement confirmed is granted.

noConfirmationRequired (39)

- **Description:** A transitional state prior to the readyForHire (40) state.
- **Conditions:** The required conditions for the hire process to reach the current state.
  - The current state is enabled (0).
  - The ELCP state is startHire.
  - The *Manager confirms new employee via workflow* GCV is set to False.

readyForHire (40)

- **Description:** No confirmation is needed. The hire process does not require a hiring manager.
- **Conditions:** The required condition for the hire process to reach the current state.
  - The current state is noConfirmationRequired (39).

directHirePending (49)

- **Description:** The new employee starts the same day as the contract starts and has access to necessary resources.

- **Conditions:** The required conditions for the hire process to reach the current state.
  - The current state is confirmed (30) or readyForHire (40).
  - The hire date is equal to first working day or first working day does not exist. Only the first eight characters of DirXML-HireDataStr and DirXML-FirstWorkingDayStr are evaluated to match the same day, but it does not use the time of day.
- **Actions:** The expected action from the components of the solution.
  - The business logic driver enables the new employee account on the hire date (DirXML-HireDateStr).

directHireComplete (50)

- **Description:** The new employee has its first working day and has access to necessary resources.
- **Conditions:** The required conditions for the hire process to reach the current state.
  - Evaluation of this state is solely triggered through the process-trigger job, which runs hourly unless it is started manually.
  - The current state is directHirePending (49).
  - The hire date is current. Only the first eight characters of DirXML-HireDataStr are evaluated to match the same day, but it does not use the time of day.
  - The object is enabled in the Identity Vault. The Login Disabled attribute is set to False.
- **Actions:** The expected action from the components of the solution.
  - The business logic driver sets employeeStatus to Active on the hire date (DirXML-HireDateStr).

earlyAccessPending (59)

- **Description:** The new employee's first working day is before the actual hire date. The new employee needs access to certain resources. The new employee might need to sign a non-disclosure agreement or other agreement to protect the company and the employee.
- **Conditions:** The required conditions for the hire process to reach the current state.
  - The current state is confirmed (30) or readyForHire (40).
  - The hire date is later than the first working day. Only the first eight characters of DirXML-HireDataStr and DirXML-FirstWorkingDayStr are evaluated to match the same day, but it does not use the time of day.
- **Actions:** The expected action from the components of the solution.
  - The business logic driver enables the account on first working day (DirXML-FirstWorkingDayStr).

earlyAccessComplete (60)

- **Description:** The new employee has the first working day and has access to necessary resources. The new employee might not be displayed in the corporate address book until the hire date is current.
- **Conditions:** The required conditions for the hire process to reach the current state.
  - The evaluation of this state is solely triggered through the process-trigger job, which runs hourly unless it is started manually.

- The current state is earlyAccessPending (59).
- The first working day is current. Only the first eight characters of DirXML-HireDataStr and DirXML-FirstWorkingDayStr are evaluated to match the same day, but it does not use the time of day.
- The object is enabled in the Identity Vault. The Login Disabled attribute is set to False.

delayedHirePending (69)

- **Description:** The new employee's hire date is after the employee has already started working.
- **Conditions:** The required condition for the hire process to reach the current state.
  - The current state is earlyAccessComplete (60).

delayedHireComplete (70)

- **Description:** The new employee's contract has started. The new employee is displayed in the corporate address book.
- **Conditions:** The required conditions for the hire process to reach the current state.
  - The evaluation of this state is solely triggered through the process-trigger job, which runs hourly unless it is manually started.
  - The current state is delayedHirePending (69).
  - The hire date is current. Only the first eight characters of DirXML-HireDataStr are evaluated to match the same day, but it does not use the time of day.
- **Actions:** The expected action from the components of the solution.
  - The business logic driver sets employeeStatus to Active on the hire date (DirXML-HireDateStr).

earlyHirePending (79)

- **Description:** The new employee's hire date is earlier than the actual first working day. The new employee does not need access to resources yet.
- **Conditions:** The required conditions for the hire process to reach the current state.
  - The current state is confirmed (30) or readyForHire (40).
  - The hire date is before the first working day. Only the first eight characters of DirXML-HireDataStr and DirXML-FirstWorkingDayStr are evaluated to match the same day, but it does not use the time of day.

earlyHireComplete (80)

- **Description:** The new employee's contract has started.
- **Conditions:** The required conditions for the hire process to reach the current state.
  - The evaluation of this state is solely triggered through the process-trigger job, which runs hourly unless it is manually started.
  - The current state is earlyHirePending (79).
  - The hire date is current. Only the first eight characters of DirXML-HireDataStr are evaluated to match the same day, but it does not use the time of day.

- **Actions:** The expected action from the components of the solution.
  - The business logic driver sets employeeStatus to Active on the hire date (DirXML-HireDateStr).

delayedAccessPending (89)

- **Description:** The employee's first working day is after the actual hire date.
- **Conditions:** The required condition for the hire process to reach the current state.
  - The current state is earlyHireComplete (80).
- **Actions:** The expected action from the components of the solution.
  - The business logic driver enables the account on the first working day (DirXML-FirstWorkingDayStr).

delayedAccessComplete (90)

- **Description:** The new employee has the first working day.
- **Conditions:** The required conditions for the hire process to reach the current state.
  - The evaluation of this state is solely triggered through the process-trigger job, which runs hourly unless it is started manually.
  - The current state is delayedAccessPending (89).
  - The first working day is current. Only the first eight characters of DirXML-FirstWorkingDayStr are evaluated to match the same day, but it does not use the time of day.
  - The object is enabled in the Identity Vault. The Login Disabled attribute is set to False.

completionPending (99)

- **Description:** The hire process is finishing.
- **Conditions:** The required condition for the hire process to reach the current state.
  - The current state is directHireComplete (50), delayedHireComplete (70), or delayedAccessComplete (90).

completed(100)

- **Description:** The hire process is finished.
- **Conditions:** The required condition for the hire process to reach the current state.
  - The current state is completionPending (99).

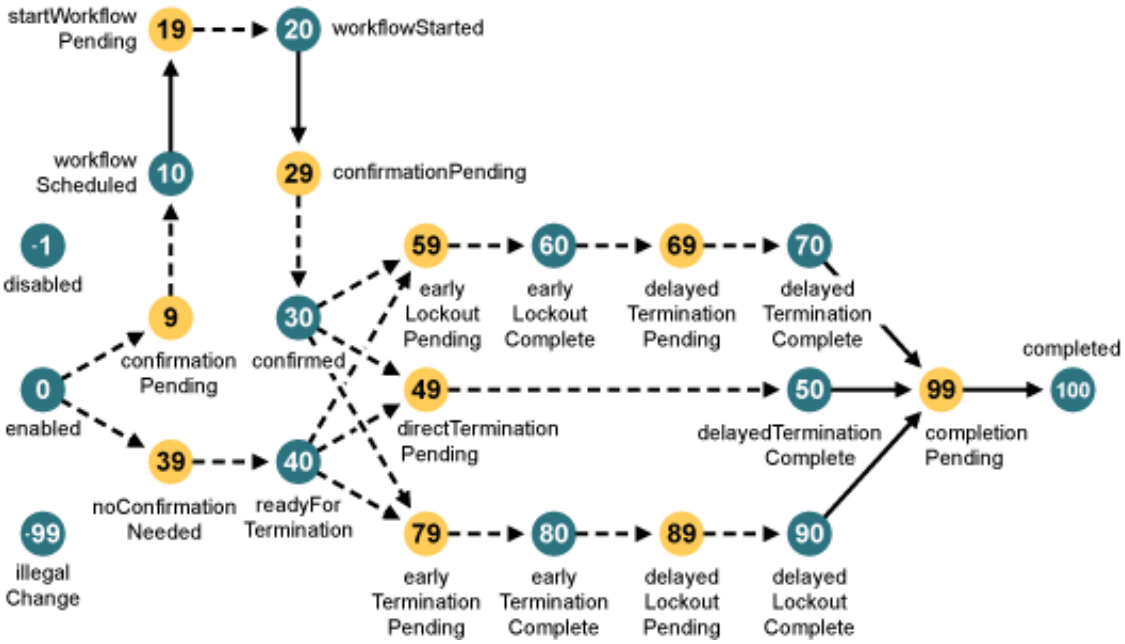
5.6 Termination Process

The section describes how the termination process works. Every step in the termination process is represented as an integer. Positive odd numbers represent requests. Positive even numbers are successes, and negative even numbers are failures. Negative odd numbers are reserved for process states and processing errors. Processes can loop, branch off into sub-process, then merge again downstream.

5.6.1 Termination Process Conditions

Figure 5-4 is graphical representation of the termination process with all of the variable conditions included.

Figure 5-4 *Termination Process*



5.6.2 The Termination Process States

This section contains a description of each state of the termination process. The termination process refers to states that occur in the employee life cycle process (ELCP) as well as to global configuration values (GCVs) that are set on the drivers. For more information on the ELCP, see Section 5.4, “Employee Life Cycle Process,” on page 32.

enabled (0)

- **Description:** The termination process is enabled.
- **Conditions:** The required conditions for the termination process to reach the current state.
  - No termination process is currently running for the object.
  - The ELCP state is hired.
  - The object has a termination date (DirXML-TerminationDateStr).
  - PID != -1

scheduleWorkflowPending (9)

- **Description:** A transitional state prior to the workflowScheduled (10) state in the termination process.
- **Conditions:** The required conditions for the termination process to reach the current state.
  - The current state is enabled (0).
  - The ELCP is at startTermination.
  - GCV *Manager confirms employee termination via workflow* is set to True.
- **Actions:** The expected actions for the components of the solution.
  - The business logic driver schedules workflows through work orders.
  - The work order driver generates work to do objects as the scheduled date becomes current.

workflowScheduled (10)

- **Description:** The workflow is scheduled to run *n* number of days before the termination date.

- **Conditions:** The required condition for the termination process to reach the current state.
  - The work order drive creates a valid work order object.
- **Actions:** The expected action from the components of the solution.
  - The business logic driver starts the workflow.

startWorkflowPending (19)

- **Description:** A transitional state to the workflowStarted (20) state.
- **Conditions:** The required condition for the termination process to reach the current state.
  - The current state is workflowScheduled (10).

workflowStarted (20)

- **Description:** The workflow is started and running.
- **Conditions:** The required condition for the termination process to reach the current state.
  - The due date on the work order object becomes current, and the WorkOrder driver deletes the work order object.

confirmationPending (29)

- **Description:** Represents a transitional state prior to the confirmed (30) state in the termination process.
- **Conditions:** The required conditions for the termination process to reach the current state.
  - The current state is workflowStarted (20).
- **Actions:** The expected actions from the components of the solution.
  - The manager claims and approves or declines the workflow.
  - If the workflow is approved, the User Application driver grants the entitlement.
  - If the workflow is declined, the User Application driver revokes the entitlement.

confirmed (30)

- **Description:** The workflow is approved. This state represents an authorized manager confirming that an employee is to be terminated and a termination date or a last working day is determined.
- **Conditions:** The required conditions for the termination process to reach the current state.
  - The current state is confirmationPending (29).
  - The entitlement is revoked.

noConfirmationRequired (39)

- **Description:** The transitional state prior to readyForTermination (40).
- **Conditions:** The required conditions for the termination process to reach the current state.
  - The current state is enabled (0).
  - The ELCP state is hired.
  - The *Manager confirms employee termination via workflow* GCV is set to False.

readyForTermination (40)

- **Description:** The termination process does not require an authorized manager's confirmation.
- **Conditions:** The required condition for the termination process to reach the current state.
  - The current state is noConfirmationRequired (39).

directTerminationPending (49)

- **Description:** The terminated employee is being dismissed and the last working day is the same as the termination date. Access to the necessary resources is denied the same day.
- **Conditions:** The required conditions for the termination process to reach the current state.
  - The current state is confirmed (30) or readyForTermination (40).
  - The termination date is equal to the last working day or the last working day does not exist. Only the first eight characters of DirXML-TerminationDateStr and DirXML-LastWorkingDayStr are evaluated to match the same day, but it does not use the time of day.
- **Actions:** The expected action from the components of the solution.
  - The business logic driver disables the employee account on the last working day (DirXML-LastWorkingDayStr).

directTerminationComplete (50)

- **Description:** The last working day is expired. The terminated employee resource access is revoked.
- **Conditions:** The required conditions for the termination process to reach the current state.
  - The evaluation of this state is solely triggered through the process-trigger job, which runs hourly unless it is started manually.
  - The current state is directTerminationPending (49).
  - The termination date is current. Only the first eight characters of DirXML-TerminationDateStr are evaluated to match the same day, but it does not use the time of day.
  - The object is enabled. The Login Disabled attribute is set to True.
- **Actions:** The expected action from the components of the solution.
  - The business logic driver sets employeeStatus to Terminated on termination date (DirXML-TerminationDateStr).

earlyLockoutPending (59)

- **Description:** The terminated employee's last working day is before the actual terminated date. The terminated employee needs access to certain resources.
- **Conditions:** The required conditions for the termination process to reach the current state.
  - The current state is confirmed (30) or readyForTermination (40).
  - The termination date is later than the last working day. Only the first eight characters of DirXML-TerminationDateStr and DirXML-LastWorkingDayStr are evaluated to match the same day, but it does not use the time of day.

- **Actions:** The expected action from the components of the solution.
  - The business logic driver disables the account on the last working day (DirXML-LastWorkingDayStr).

earlyLockoutComplete (60)

- **Description:** The terminated employee's last working day occurs and access to resources is revoked. The terminated employee might be displayed in the corporate address book until the termination date is current.
- **Conditions:** The required conditions for the termination process to reach the current state.
  - The evaluation of this state is solely triggered through the process-trigger job, which runs hourly unless it is manually started.
  - The current state is earlyLockoutPending (59).
  - The last working day is current. Only the first eight characters of DirXML-LastWorkingDayStr are evaluated to match the same day, but it does not use the time of day.
  - The object is enabled. The Login Disabled attribute is set to True.

delayedTerminationPending (69)

- **Description:** The employee's termination date is after the employee has already stopped working.
- **Conditions:** The required conditions for the termination process to reach the current state.
  - The current state is earlyLockoutComplete (60).

delayedTerminationComplete (70)

- **Description:** The terminated employee's contract ended. The employee is removed from the corporate address book.
- **Conditions:** The required conditions for the termination process to reach the current state.
  - The evaluation of this state is solely through the process-trigger job, which runs hourly unless it is manually started.
  - The current state is delayedTerminationPending (69).
  - The termination date is current. Only the first eight characters of DirXML-TerminationDateStr are evaluated to match the same day, but it does not use the time of day.
- **Actions:** The expected action from the components of the solution.
  - The business logic driver sets employeeStatus to terminated on the termination date (DirXML-TerminationDateStr).

earlyTerminationPending (79)

- **Description:** The employee's termination date is earlier than the actual last working day. The employee still needs access to resources.

- **Conditions:** The required conditions for the termination process to reach the current state.
  - The current state is confirmed (30) or readyForTermination (40).
  - The termination date is before the last working day. Only the first eight characters of DirXML-TerminationDateStr and DirXML-LastWorkingDayStr are evaluated to match the same day, but it does not use the time of day.

earlyTerminationComplete (80)

- **Description:** The terminated employee's contract ends.
- **Conditions:** The required conditions for the termination process to reach the current state.
  - The evaluation of this state is solely triggered through the process-trigger job, which runs hourly unless it is manually started.
  - The current state is earlyTerminationPending (79).
  - The termination date is current. Only the first eight characters of DirXML-TerminationDateStr are evaluated to match the same day, but it does not use the time of day.
- **Actions:** The expected action from the components of the solution.
  - The business logic driver sets employeeStatus to Terminated on the termination date (DirXML-TerminationDateStr).

delayedLockoutPending (89)

- **Description:** The terminated employee's last working day is after the actual termination date.
- **Conditions:** The required condition for the termination process to reach the current state.
  - The current state is earlyTerminationComplete (80).
- **Actions:** The expected action from the components of the solution.
  - The business logic driver disables the account on the last working day (DirXML-LastWorkingDayStr).

delayedLockoutComplete (90)

- **Description:** It is the last working day for the employee.
- **Conditions:** The required conditions for the termination process to reach the current state.
  - The evaluation of this state is solely triggered through the process-trigger job, which runs hourly unless it is manually started.
  - The current state is delayedLockoutPending (89).
  - The last working day is current. Only the first eight characters of DirXML-LastWorkingDayStr are evaluated to match the same day, but it does not use the time of day.
  - The object is enabled. The Login Disabled is set to True.

completionPending (99)

- **Description:** The ELCP is finishing.

- **Conditions:** The required condition for the termination process to reach the current state.
  - The current ELCP state is directTerminationComplete, delayedTerminationComplete, or delayedLockoutComplete.

completed (100)

- **Description:** The ELCP is finished.
- **Conditions:** The required condition for the termination process to reach the current state.
  - The current ELCP state is completionPending.

Object Synchronization

6

Most configuration of solutions can be done through global configuration values (GCVs) and filters. This makes the base installation easy to deploy as because using a minimal number of pages for values and setting authoritative sources provides a rapid deployment.

The basic configuration of each driver follows these guidelines or explicitly documents differences:

6.1 Attributes

The attributes that are synchronized are included in the filters for each driver.

6.2 Passwords

The Resource Kit supports bidirectional user password synchronization to all systems that support it.

Drivers

7

The Resource Kit contains two types of drivers:

7.1 Application Drivers

An application driver synchronizes data between the Identity Vault and the applications.

- **Active Directory Driver:** User accounts and passwords are synchronized between the Identity Vault and Active Directory.
- **Notes Driver:** User accounts are synchronized between the Identity Vault and the Notes database.
- **User Import/Export Driver:** The user import/export driver adds users objects to the Identity Vault. This is a Delimited text driver. It uses Java classes to extend the functionality of the base Delimited Text driver. For more information, see “Delimited Text Driver Extensions” in the *Identity Manager 3.6 Driver for Delimited Text Implementation Guide*.
- **Image Import/Export Driver:** The image import/export drivers adds images to the user objects. This is a Delimited Text driver. It uses Java classes to extend the functionality of the base Delimited Text driver. For more information, see “Delimited Text Driver Extensions” in the *Identity Manager 3.6 Driver for Delimited Text Implementation Guide*.

7.2 Service Drivers

A service agent driver is a driver that provides a special service instead of synchronizing data between the Identity Vault and an Application. Service drivers can be implemented as loopback drivers, or other drivers such as the Entitlement Service driver, which is responsible for making Roles-Based Entitlements work.

- **Business Logic Driver:** The Business Logic driver creates the full name from the first, middle, and last name. It also uses a GCV that allows the following tuning:
  - First Middle Last
  - Last First Middle
  - First Last
  - Last First
- **Role Based Entitlements Service Driver:** Grants and revokes entitlements based on roles.
- **User Application Driver:** The User Application driver enables workflows to do the following:
  - Refresh the user application if configuration changes occur in the Identity Vault
  - Start workflows based on certain conditions in the Identity Vault
  - Update workflow status based on entitlement result

- **ID Provider Driver:** Assigns unique IDs for all of the accounts. The ID assignment comes from the driver, not from the Identity Vault or any connected application. For more information about the driver, see the *Identity Manager 3.6 ID Provider Driver Implementation Guide*.
- **State Machine Driver:** The State Machine driver tracks the status of each account as it goes through the employee life cycle process. It is required to track the status of each account.